COMPUTRONIC
TI994A
Software
fur
Heim
computer

Unser Kurs

Programmieren in Forth Teil 1

F. Brall

Literatur:

Zech Ronald: Die Programmiersprache Forth
Brodie Leo: Starting Forth
E. Floegel: Forth Handbuch
Zimmer Tom: Hes Ware 64 Forth Manual

Vorstellung der Programmiersprache

Wer schon eine Zeit in Basic programmiert hat, wird wissen,dass es hin und wieder Sachen gibt, die nicht oder nur recht schwierig zu prorammieren sind.Fuer solche Faelle hat man bei grossen Rechnern die Moeglichkeit auf andere Sprachen zurueckgreifen zu koennen.Es gibt FORTRAN fuer mathematische Problemloesungen, COBOL fuer kommerzielle Zwecke, ASSEMBLER fuer zeitkritische Aufgaben,Basic fuer die allgemeinen Problemloessungen auf Microcomputern und,und, und.
FORTH, das heisst soviel wie "Die Vierte" ,gehoert zu der juengsten Generation von Programmiersprachen, wie der Name schon sagt, zur vierten Generation.Man hat also versucht alle Vorteile von den bekannten, renomierten Programmiersprachen in Forth zu implementieren, ohne aber deren Fehler mit zu uebernehmen.
FORTH hat, schon in seiner Struktur, einige sehr markante Vorteile, die besonders bei den Microcomputern von Vorteil sind:

1) FORTH ist auf fast allen Microcomputern implementierbar und benoetigt nur wenig Speicherplatz
2) FORTH Programme sind wesentlich kuerzer als BASIC und ASSEMBLER Programme
3) FORTH ist in der Regel 10 mal schneller als BASIC
4) FORTH kann auch mit Kassettenrecorder betrieben werden.Vorteilhaft ist ein Floppy Laufwerk natuerlich in jedem Fall.
5) FORTH ist problemlos erweiterbar
6) FORTH-"Hochsprache" und FORTH-"Assembler" laesst sich beliebig mischen.
7) FORTH erlaubt strukturierte Programmierung

Forth ist urspruenglich fuer die Programmierung von Steuerungsaufgaben entwickelt wurden.Da die Sprache jedoch sehr leicht erweiterbar ist, kann man sie fuer fast alle Anwendungen verwenden.Durch das implementieren einiger Grafik-Befehle wird aus der Prozesssprache schnell eine geeignete Sprache fuer Spiele und Grafik-Anwendungen.Jeder kann FORTH um die Befehle erweitern, die er fuer seine Anwendungen gerade benoetigt.Trotz zahlreicher Erweiterungen sind FORTH Programme sehr leicht auf einen anderen Rechner zu uebertragen.Fehlende Befehle (WORTE) koennen einfach noch implementiert werden.

Ich hoffe mir ist es gelungen einige Vorteile von FORTH darzustellen.In den folgenden Teilen dieses Kurses werden Sie schrittweise mit den Eigenheiten der Sprache vertraut gemacht. ..... beabsichtigt ebenfalls regelmaessig Forth-Programme zu veroeffendlichen.Es ist schade das bisher kaum eine Zeitschrift naeher auf Forth eingegangen ist ,obwohl es sicher eine der leistungsfaehigsten Sprachen fuer Microcomputer ist und immer mehr Freunde findet.

Unser Kurs

Einleitung

FORTH ist ein vollstaendiges selbstaendiges Sprachsystem, das sowohl Compiler als auch Interpreter ist und sein eigenes Betriebssystem beinhaltet.Das besondere an FORTH ist das in erster Linie mit dem "Stapel" gearbeitet wird.Wer schon einmal in Assembler programmiert hat kennt die Arbeitsweise des Stapels.Der Stapel arbeitet nach dem LIFO-System (letzte Eingabe erste Ausgabe).Dies bedeutet das auf ihm etwas abgelegt werden kann und vieleicht spaeter wieder heruntergenommen.Forth besitzt zwei Stapel den Daten- und den Returnstapel.Auf die Funktion wird spaeter noch eingegangen.
Die verwendung eines Stapels waere an fuer sich nichts Neues.Jeder HP-Taschenrechnen verwendet einen Stapel zum Speichern von Zahlen.Das andersartige an FORTH ist das Woerterbuch welches auch als Stapel ausgelegt ist.
Die Sprache besteht aus Worten, die nach ihrer Definition, das heisst nach der Festlegung dessen, was das Wort bewirken soll, auf dem Definitionsstapel, dem Woerterbuch abgelegt werden.Bei der Definition eines Wortes koennen schon vorherige Definitionen verwendet werden.Ein Forth-Programm besteht also in der Regel aus zahlreichen Definitionen die sich gegenseitig aufrufen.Der Start eines Programmes besteht dann nur im Aufruf eines Wortes.
Um Woerter wieder zu loeschen benutzt man den Befehl FORGET .
Der Aufruf FORGET TEST loescht alle Worte, beginnend bei TEST bis zum obersten Eintrag in den Stapel.Das loeschen eines einzigen Wortes im Stapel ist nicht moeglich und waere auch nicht sinnvoll.Denn spaetere Definitionen koennten dieses Wort benutzen und waeren somit nicht mehr moeglich.Es gibt auch die Moeglichkeit ein Wort mehrmals zu definieren.In diesem Fall wird immer das zuletzt definierte Wort ausgefuert.Welchen nutzen dies hat wird in einer der naechsten Folgen besprochen.
Berechnungen erfolgen in Forth ueber den Datenstapel den wir auch als STACK bezeichnen koennen.Da alle Operanden ueber den Stack geleitet werden ergibt sich die etwas ungewohnte Rechenschreibweise UPN (Umgekehrte Polnische Notation).Auch die schon genannten HP-Taschenrechner verwenden diese Art der Eingabe.Um beispielsweise die Zahlen 5 und 7 zu addieren muss folgendes eingegeben werden:

in FORTH 5 7 + in BASIC 5 + 7

Nach der Eingabe dieses Beispiels werden die beiden Summanden vom Stack geholt und addiert und das Ergebnis wieder auf den Stack geschrieben.Man kann dies leicht ueberpruefen indem man sich das Ergebnis durch den Befehl "." ausgeben laesst.Der Befehl "." holt die oberste Zahl vom Stack und gibt sie auf dem Schirm aus.Ein nochmaliges Aufrufen des Befehles "." bewirkt eine Fehlermeldung dar der Stack keine Zahl mehr beinhaltet.
Wichtig ist das die einzelnen Worte und Zahlen immer durch mindestens 1 Leerzeichen getrennt werden, da der Computer sonst das Wort nicht erkennen kann.Nach erfolgreicher Ausfuehrung eines Wortes meldet sich der Computer mit "OK", dies entspricht in BASIC der "READY" Meldung.
FORTH ist eine sehr nachsichtige Sprache.Alles, was eingegeben und nicht direkt als falsch erkannt wird, wird angenommen und eventuell sogar ins Woerterbuch eingetragen.Man darf also nicht blindlings darauf los programmieren, sondern man sollte sich einigermassen ueberlegen, was und wie man es programmiert.

Unser Kurs

Bei der Beschreibung des Woerterbuches und der Operationen mit dem Stapelspeicher (Stack) werden Bezeichnungen verwendet, wie sie im Installation Manual fuer FORTH festgelegt sind.Diese Bezeichnungen sollten Sie sich einpraegen da sie auch in den kommenden Folgen dieses Kurses immer wieder verwendet werden.

Diese Bezeichnungen sind:

adr	Speicheradresse
b	8 bit Byte (die oberen 8 Bit sind Null)
c	7 bit ASCII-Zeichen
d	32 bit doppelt lange Festkommazahl mit Vorzeichen
f	Bool'sche Variable Ø= "false" (falsch)
n	16 bit Zahl mit Vorzeichen
u	16 bit Zahl ohne Vorzeichen
<RETURN>	RETURN Taste

Bei der Beschreibung eines Wortes wird kuenftig folgendes Schema angewendet:

WORT <RETURN> Anzeige des Computers (Stapel Vorher --> Stapel Nachher)

KOMMENTAR

Beispiel:

. <RETURN> OK (n1 n2 n3 --> n1 n2)

n3 wird auf dem Bildschirm ausgegeben

Wie Sie schon wissen ist "." eine Forth Anweisung die die oberste Zahl des Stapels auf dem Schirm ausgibt.Dies ist also die Print-Anweisung.Vor der Ausfuehrung des Befehles befinden sich 3 Zahlen n1,n2,n3 im Stapelspeicher.Davon ist die am weitesten rechts stehende Zahl im TOS (Top of Stack).In diesem Fall also n3.Nach der Ausfuehrung des Befehles sind nur noch zwei Zahlen (n1,n2) im Stapel wobei nun n2 an oberster Stelle steht.

Um noch einmal die Funktion des Stapels zu verdeutlichen geben Sie bitte folgendes Beispiel ein:

1 2 3 4 5 6 <RETURN> OK (empty --> n1 n2 n3 n4 n5 n6)

Die Zahlen 1 bis 6 wurden nacheinander auf den Stapel gelegt.Um sie wieder abzurufen geben Sie folgendes ein:

. <RETURN> 6 OK (n1 n2 n3 n4 n5 n6 --> n1 n2 n3 n4 n5)
. <RETURN> 5 OK (n1 n2 n3 n4 n5 --> n1 n2 n3 n4)
. <RETURN> 4 OK (n1 n2 n3 n4 --> n1 n2 n3)
. <RETURN> 3 OK (n1 n2 n3 --> n1 n2)
. <RETURN> 2 OK (n1 n2 --> n1)
. <RETURN> 1 OK (n1 --> empty)

Wie aus dem Beispiel zu erkennen ist,werden die Zahlen in der umgekehrten Reihenfolge ausgegeben.Natuerlich ist es gleich ob man den "." Befehl einzeln mit RETURN quitiert oder nur durch ein Leerzeichen trennt.

. . . . . <RETURN> 6 5 4 3 2 1 OK

Mit dieser Einführung in das STACK-Konzept möchten wir Teil 1 abschließen. Im nächsten Heft stehen die Arithmetik-Operationen im Vordergrund.

Programmieren in Forth Teil 2

F. Brall

Literatur:

Zech Ronald: Die Programmiersprache Forth
Brodie Leo: Starting Forth
E. Floegel: Forth Handbuch
Zimmer Tom: Hes Ware 64 Forth Manual

Arithmetik-Operationen

Die 4 Grundrechenarten sind in FORTH und BASIC durch die gleichen Bezeichnungen gekennzeichnet.

+	[n1 n2 – –> Summe]	Addition
–	[n1 n2 – –> Differenz]	Substraktion
*	[n1 n2 – –> Produkt]	Multiplikation
/	[n1 n2 – –> Quotient]	Division

Wie bei allen FORTH-Wörtern, werden auch hier die Operanten auf dem Stack erwartet, und nach der Ausführung durch das Ergebnis überschrieben.
Einige Beispiele verdeutlichen die Arbeitsweise der Befehlswörter:

12 3	⟨RETURN⟩ OK	[empty – –> n1 n2]
+	⟨RETURN⟩ OK	[n1 n2 – –> Summe]
.	⟨RETURN⟩ 15 OK	[Summe – –> empty]

Eine andere Schreibweise:
12 3 + . ⟨RETURN⟩ 15 OK
43 213 + . ⟨RETURN⟩ 256 OK
1000 344 + . ⟨RETURN⟩ 1344 OK
500 100 50 + + . ⟨RETURN⟩ 650 OK

Die Aufgaben 14 – 4, 12 * 3 und 2000 / 50 werden in FORTH wie folgt gelöst:
14 4 – . ⟨RETURN⟩ 10 OK
12 3 * . ⟨RETURN⟩ 36 OK
2000 50 / . ⟨RETURN⟩ 40 OK

Alle oben aufgeführten Wörter, sind INTEGER-Operationen. Dies bedeutet, daß die Operanten sowie das Ergebnis intern, 16 Bit belegen. Für die Anwendung ergibt sich somit folgender Zahlenbereich:
–32768 <– – – – – – – – – – – 0 – – – – – – – – – – – –> 32767
Dieser Zahlenbereich darf nicht überschritten werden, da sich sonst das Ergebnis entsprechend verfälscht. Gleitkommazahlen sind in FORTH unzulässig.

Folgende Beispiele zeigen die Grenzen der Integer-Arithmetik:

10000 4 * .	⟨RETURN⟩ –25536 OK	Richtig wäre 10000
32767 1 + .	⟨RETURN⟩ –32768 OK	Richtig wäre 32768
5 2 / .	⟨RETURN⟩ 2 OK	Richtig wäre 2.5
11 2 / .	⟨RETURN⟩ 5 OK	Richtig wäre 5.5

Einige Basic-Freunde werden sich jetzt fragen: »Was soll ich mit einer Programmiersprache die nicht einmal richtig 5 durch 2 teilen kann«? Diese Frage ist sicher nicht ganz unberechtigt, jedoch sollte man bedenken, daß in sehr vielen Spiel-, Grafik-, Steuerungs- und Rechenprogrammen eine Integer-Arithmetik völlig ausreicht. Eine Gleitkommazahl können wir durch Wahl einer anderen Einheit in eine Integer-Zahl umwandeln. Beispielsweise können wir den Betrag 1,25 DM in 125 Pf umwandeln.
In einer der nächsten Folgen werden noch einige 32 Bit Arithmetik-Befehle vorgestellt. Diese erlauben durch geschickte Programmierung eine höhere Genauigkeit, als sie in Basic vorhanden ist.

Die FORTH-Schreibweise verschiedener Rechenaufgaben:
Aufgabe: 4 + [17 * 12]
17 12 * 4 + . ⟨RETURN⟩ 208 OK
Durch das Stack-Konzept werden Klammern überflüssig.

Aufgabe: [3 + 9] * [4 + 6]
3 9 + 4 6 + * . ⟨RETURN⟩ 120 OK

Aufgabe: 45 / 5 * 100
45 5 / 100 * . ⟨RETURN⟩ 900 OK

Weitere Arithmetik Befehle sind:

MOD [u1 u2 – –> Rest]
Dividiere u1 durch u2 und lege Rest auf Stack
11 3 MOD . ⟨RETURN⟩ 2 OK
8 10 MOD . ⟨RETURN⟩ 8 OK

/MOD [u1 u2 – –> Rest Quotient]
Dividiere u1 durch u2 und lege Rest sowie Quotient auf Stack
11 3 /MOD . ⟨RETURN⟩ 3 2 OK
8 10 /MOD . ⟨RETURN⟩ 0 8 OK

*/ [n1 n2 n3 – –> n4]
Errechne den Verhältnis-Faktor n4 = n1 * n2/n3. Das Produkt n1 * n2, ist als Zwischenergebnis 32 Bit lang. Hierdurch ist eine höhere Rechengenauigkeit gewährleistet, als durch die Sequenz n1 n2 * n3 / erreichbar wäre.
Diese Anweisung wird häufig zum Rechnen mit genauen Konstanten verwendet. Folgende Konstanten können durch das Verhältnis n2/n3 errechnet werden:

Konstante	Richtiger Wert	Verhältnis n2/n3
PI	3.1415926	355 / 113
Wurzel aus 2	1.4142135	19601 / 13860
Wurzel aus 3	1.7320508	18817 / 10864

Einige Anwendungsbeispiele:
10 355 113 */ . ⟨RETURN⟩ 31 OK Entspricht 10 * PI
100 355 113 */ . ⟨RETURN⟩ 314 OK Entspricht 100 * PI

Wie aus dem Beispiel zu erkennen ist, wird mit diesem Befehl die größte Genauigkeit, die Integer-Zahlen bieten, erzielt.

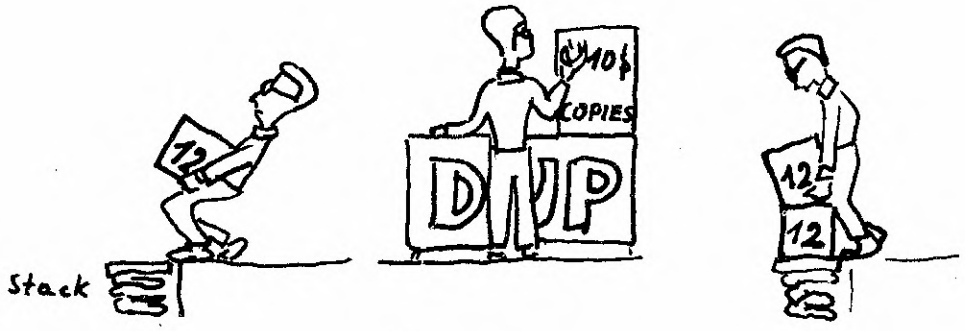
*/MOD [n1 n2 n3 – –> Rest n4]
Dieser Befehl entspricht dem Vorangehenden, mit der Ausnahme, daß auch der Rest der Division auf den Stack gelegt wird.
10 355 113 */MOD . . ⟨RETURN⟩ 31 47 OK
100 355 113 */MOD . . ⟨RETURN⟩ 314 18 OK

ABS [n1 – –> u1]
Dieser Befehl wandelt n1 mit Vorzeichen in die vorzeichenlose Zahl u1 um. In Basic gibt es den gleichen Befehl.
– 123 ABS . ⟨RETURN⟩ 123 OK
– 1 ABS . ⟨RETURN⟩ 1 OK
5 ABS . ⟨RETURN⟩ 5 OK

Häufig benutzte Befehls-Folgen, wurden in FORTH zu einem Befehl zusammengefügt:
1 + [n1 – –> n2] entspricht der Befehlsfolge 1 +
1 – [n1 – –> n2] entspricht der Befehlsfolge 1 –
2 + [n1 – –> n2] entspricht der Befehlsfolge 2 +
2 – [n1 – –> n2] entspricht der Befehlsfolge 2 –
2 * [n1 – –> n2] entspricht der Befehlsfolge 2 *
2/ [n1 – –> n2] entspricht der Befehlsfolge 2 /
Diese Befehls-Kombinationen sind nicht in allen FORTH-Versionen enthalten, können jedoch sehr leicht definiert werden. Das Definieren von Befehlen wird Thema des nächsten Teiles werden.

Stack Manipulation

Wie wir schon bei einigen Befehlen bemerkt haben, ist es wichtig, daß die Parameter eines Befehls immer in der richtigen Reihenfolge auf dem Stack liegen. Bei umfangreichen Befehlsfolgen, muß deshalb oft die Reihenfolge der Operanten geändert werden, um eine anschließende Operation zu ermöglichen. Um dies zu erreichen, werden folgende Befehle verwendet:
[n1 – –> n1 n1]
DUP ist der am häufigsten verwendete Befehl. Man kann ihn sich sehr leicht merken, da DUP für Duplizierung steht. Er hat eine einfache, jedoch wichtige Aufgabe. Dieser Befehl dupliziert die oberste Zahl des Stack's.
10 0 DUP . . . ⟨RETURN⟩ 0 0 10 OK
12 DUP . . ⟨RETURN⟩ 12 12 OK



–DUP [n1 – –> n1 n1]

Dieser Befehl entspricht dem DUP Befehl, mit der Ausnahme, daß zuvor getestet wird, ob die zu duplizierende Zahl von Null verschieden ist. Nur wenn dies der Fall ist, erfolgt ihre Duplizierung.

10 0 –DUP . . ⟨RETURN⟩ 0 10 OK
12 –DUP . . ⟨RETURN⟩ 12 12 OK

DROP [n1 – –> empty]

ist das einfachste Befehlwort, welches lediglich die oberste Zahl des Stack's löscht.

10 0 DROP . ⟨RETURN⟩ 10 OK
12 DROP . ⟨RETURN⟩ Fehlermeldung: Stack leer

SWAP [n1 n2 – –> n2 n1]

Dieser Befehl verändert ganz einfach die Anordnung der beiden obersten Zahlen. Dies ist oft vor Arithmetik-Operationen erforderlich.

10 0 SWAP . . ⟨RETURN⟩ 10 0 OK
33 4 SWAP . . ⟨RETURN⟩ 33 4 OK

OVER [n1 n2 – –> n1 n2 n1]

Dieser Befehl arbeitet ähnlich wie der DUP Befehl, jedoch wird hier der »Second« (d. h. die vorletzte Eintragung in den Stack) kopiert.

10 0 OVER . . ⟨RETURN⟩ 10 0 10 OK
33 4 OVER . . ⟨RETURN⟩ 33 4 33 OK

ROT [n1 n2 n3 – –> n2 n3 n1]

Hiermit wird die Anordnung der höchsten drei Stackeintragungen in zyklischer Rotation verändert. Dabei landet der älteste Eintrag an oberster Stelle.

10 20 30 ROT . . . ⟨RETURN⟩ 10 30 20 OK
10 20 30 ROT ROT . . . ⟨RETURN⟩ 20 10 30 OK
10 20 30 ROT ROT ROT . . . ⟨RETURN⟩ 30 20 10 OK

Einige erweiterte FORTH-Versionen, wie 64 FORTH oder FORTH 79, besitzen noch einige, zusätzliche Stack-Manipulations Befehle. Einen gewissen Standard besitzen folgende Befehle:

DEPTH [empty – –> n1]

Es wird die Anzahl der auf dem Stack liegenden Parameter übergeben.

10 20 DEPTH . . . ⟨RETURN⟩ 2 20 10 OK
DEPTH . ⟨RETURN⟩ 0 OK

PICK [n1 – –> n2]

Die »n-te« Zahl wird auf den Stack gelegt. 1 PICK entspricht DUP; 2 PICK entspricht OVER

12 4 34 33 4 PICK . . . . . ⟨RETURN⟩ 12 33 34 4 12 OK
12 3 2 PICK . . . ⟨RETURN⟩ 12 3 12 OK

ROLL [n – –>]

Dieser Befehl rotiert die »n-te« Zahl des Stack's an die oberste Stelle. 2 ROLL entspricht SWAP; 3 ROLL entspricht ROT.

12 13 14 15 4 ROLL . . . . ⟨RETURN⟩ 12 15 14 13
12 13 14 15 3 ROLL . . . . ⟨RETURN⟩ 13 15 14 12

Programmieren in Forth

ISL 48K SPECTRUM ISL

SPLAT!
ARCADE

ONE OF THE MOST ORIGINAL AND COMPELLING ARCADE GAMES EVER PRODUCED! STARRING ZIPPY!!

'SPLAT! is one of the most addictive games I have ever played on the 48K SPECTRUM. It is certainly the most original!'
Computer & Video Games

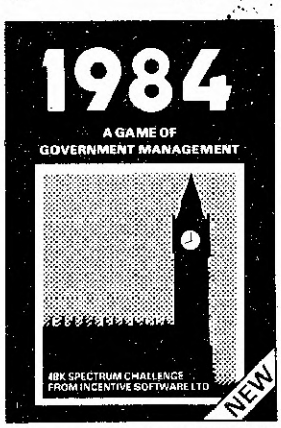
NOW AVAILABLE FROM WH SMITH AND BOOTS.
N001 £5.50

MOUNTAINS OF KET
ADVENTURE

A MONSTER OF AN ADVENTURE PROGRAM! COMBAT INTER ACTIVE BEINGS, MONETARY SYSTEM MAGIC, EDGAR, SAVE LOAD FACILITY PLUS MANY OTHER FEATURES

As well as being a fast ingenious compelling adventure in itself the Mountains of Ket is the first of a 3 part series that builds into a mammoth adventurers challenge

Incentive. It could be advantageous if you achieve 100%!!
N002 £5.50



1984
THE GAME OF ECONOMIC SURVIVAL

THE BRITISH ECONOMY WITH YOU AT THE CONTROLS! WHAT SORT OF CHANCELLOR WOULD YOU MAKE WITH SEVERAL BILLION POUNDS TO SPEND & FIVE YEARS TO THE NEXT GENERAL ELECTION? GRAPHIC DISPLAYS, HISTOGRAMS & AN ANNUAL PERFORMANCE RATING ARE ALL INCLUDED TO SHOW HOW YOU ARE DOING. HOW MANY YEARS WILL YOU LAST?

FREE INSIDE Pocket Guide to Running Britain!
N003 £5.50

All programs run in the **48K ZX SPECTRUM** and are available from all good computer shops. In case of difficulty please order direct using the coupon below.

Please send me (tick boxes) required)
SPLAT ☐ MOUNTAINS OF KET ☐ 1984 ☐
All at £5.50 each (inclusive of VAT and 1st class postage)
I enclose cheque/P.O. for £ or debit my Access Account No
Name
Address

ISL INCENTIVE SOFTWARE LTD., 54 London Street, Reading RG1 4SQ. Tel: Reading (0734) 591678

Definition neuer Befehls-Worte Teil 3

Nachdem wir uns im letzten Teil mit den Arithmetik-Operationen beschäfitgt haben, wollen wir uns nun mit der Definition neuer Befehle befassen. Die bislang vorgestellten FORTH-Befehle sind bis jetzt ausschließlich interaktiv und interpretativ aufgerufen worden. Das bedeutet, daß ein Wort (z. B. DUP oder +) mit der Eingabe seines Namens in die Tastatur aufgerufen wird, indem der äußere Interpreter dann das Wörterbuch (Dictionary) nach diesem Wort absucht, es lokalisiert und ausführt. Diese Art der Befehls-Abarbeitung wird im Gegensatz zu Basic nur bei der direkten Eingabe von Befehlen verwendet. Wir haben nun bereits so viele Befehle kennengelernt, daß wir daran gehen können, kleinere Programme damit zu erstellen. Wie schon im ersten Teil erwähnt, ist das gleichbedeutend damit, die zu anfangs erwähnten Befehle zu neuen Befehlen zusammenzusetzen.

Wollen wir einen Befehl definieren, so müssen wir als erstes mit dem Befehl »:«, den FORTH-COMPILER einschalten. Als nächstes wird die Bezeichnung und danach die eigentliche Definition des neuen Befehles eingegeben. Mit dem Befehl »;« wird der FORTH-COMPILER wieder ausgeschaltet und damit die Definition abgeschlossen.

Wir wollen nun einen Befehl definieren, welcher eine Zahl mit sich selbst multipliziert und das Ergebnis auf den Stack legt. Als Bezeichnung wählen wir »QUADRO«.

```
: QUATRO DUP * ; ⟨RETURN⟩ OK
```

Wie zu erkennen ist, besteht die eigentliche Definition aus einer unbegrenzten Zahl von Befehlen. Der Befehl DUP dupliziert die oberste Zahl des Stack und der nachfolgende »*« Befehl bildet, durch die Multiplikation, die Quadratzahl.

In den nachfolgenden Beispielen können wir den neuen Befehl testen:

```
 2 QUATRO · ⟨RETURN⟩ 4 OK
 5 QUATRO · ⟨RETURN⟩ 25 OK
10 QUATRO · ⟨RETURN⟩ 100 OK
```

Durch den Befehl VLIST können wir das gesamte WÖRTERBUCH (BEFEHL's-Liste) auf Drucker oder Monitor ausgeben lassen. Der neue Befehl QUATRO steht nun am Anfang dieser Liste.

Bei der Defination von neuen Befehlen muß darauf geachtet werden, daß die Befehlsbezeichnung keine 31 Zeichen übersteigt. Es ist nicht unbedingt notwendig und auch nicht immer möglich, daß die Defination in eine einzige Zeile geschrieben wird. Als Beispiel wollen wir einen Befehl definieren, welcher die Kubikzahl eines Operanten errechnet:

```
: KUBIK DUP DUP ** ; ⟨RETURN⟩ OK
```

Wir können das gleiche auch wie folgt eingeben:

```
: KUBIK      ⟨RETURN⟩
    DUP DUP  ⟨RETURN⟩
    **       ⟨RETURN⟩
    ;        ⟨RETURN⟩ OK
```

Bei der direkten Eingabe von Befehlen ist die Schreibweise kaum von Bedeutung, da die Struktur der Eingabe nicht gespeichert wird. Ganz anders ist dies bei der Programmierung mit dem Texteditor. Hier sollte man unbedingt auf Strukturierung und gute Dokumentation achten, um auch später noch Erweiterungen oder Änderungen vornehmen zu können. Wir werden uns später noch genauer mit diesem Thema auseinander setzen.

Nun wollen wir auch unseren KUBIK-Befehl testen:

```
 2 KUBIK . ⟨RETURN⟩ 8 OK
 5 KUBIK . ⟨RETURN⟩125 OK
10 KUBIK . ⟨RETURN⟩1000 OK
```

Werden in einem Programm oft die gleichen Befehls-Folgen verwendet, so ist es sinnvoll diese zu einem Befehl zusammenzufassen:

```
: 1+ 1+ ; ⟨RETURN⟩ OK
: 1- 1- ; ⟨RETURN⟩
: 2+ 2+ ; ⟨RETURN⟩
```

Einige dieser einfachen Befehls-Folgen sind im FORTH-Standart schon enthalten und wurden im letzten T eil schon vorgestellt. Es ist nun sehr leicht, die fehlenden Kombinationen zu definieren. Wird in FORTH ein Befehl zweimal definiert, so gilt immer die letzte Defination.

FORTH befehle können auch zur Umsetzung dienen. Bei den meisten farbigen Computern dient ein Zahlencode zur Bestimmung einer Farbe.

Farbcode des Commodore 64: 0 = Schwarz
1 = Weiß
2 = Rot
3 = Türkis
usw.

Wollen wir statt des umständlichen Codes einfach die Farbe eingeben, können wir dies durch die folgenden Definationen erreichen:

```
: SCHWARZ 0 ;  ⟨RETURN⟩
: WEISS   1 ;  ⟨RETURN⟩
: ROT     2 ;  ⟨RETURN⟩
: TÜRKIS  3    ⟨RETURN⟩
usw.
```

Die Funktion ist sehr einfach: Nach Eingabe der Farbbezeichnung legt der entsprechende Befehl den dazugehörigen Wert auf den Stack.

Es ist auch möglich mit den Farben zu rechnen:

```
WEISS ROT + .  ⟨RETURN⟩ 3 OK
```

Mit Hilfe eines PRINT Befehls können wir auch leicht eine Vokabel-Übersetzung realisieren. Der PRINT Befehl besitzt in FORTH folgende Syntax: ."text".

Elementare String-Befehle

Leider existiert in FORTH nur eine kleiner Befehlssatz zu Stringverarbeitung. Der FORTH-Kern enthält jedoch einige Worte für einfache Ein- und Ausgaben von Strings (Texten), die für viele Applikationen völlig ausreichen dürften.

CR

Dieser Befehl bewirkt auf dem Ausgabegerät (normal Monitor) Zeilenvorschub und Wagenrücklauf.

12 13 14 . . CR . (RETURN) 14 13
12 OK
. "TRONIC· CR . „VERLAG" (RETURN) TRONIC
Verlag OK

SPACE

Ein SPACE-Zeichen wird ausgegeben
. „APRIL" SPACE . „MAI" (RETURN)
APRIL MAI OK

Spaces (n1 → empty)

Eine bestimmte Anzahl von SPACE-Zeichen wird ausgegeben.
. „APRIL" 4 SPACE . „MAI" (RETURN) APRIL MAI OK

EMIT (c → empty)

Der ASCII-Charakter c wird ausgegeben. In Basic lautet dieser Befehl CHR$ (x) .
65 EMIT (RETURN) A OK
66 EMIT (RETURN) B OK

EXPECT (adr n → empty)

Dieser Befehl entspricht in etwa dem INPUT des Basic-Interpreters. Ein String mit ‚n'-Zeichen wird von der Tastatur erwartet und ab Adresse ‚adr' abgespeichert. Ein RETURN kann diese Prozedur beenden, auch wenn die Anzahl ‚n' noch nicht erreicht sein sollte.
10000 11 EXPECT (RETURN)
COMPUTRONIC (RETURN) OK

TYPE (adr n → empty)

Dieser Befehl gibt eine Zeichenkette, die ab Adresse ‚adr' gespeichert und ‚n'-Zeichen lang ist, auf dem Ausgabegerät aus.
10000 11 TYPE (RETURN) COMPUTRONIC OK

COUNT (adr1 → adr2 n)

Dieser Befehl erwartet eine Adresse ‚adr 1' auf dem Stack und ermittelt hieraus die Startadresse und die Zeichenzahl des Strings. Dies sind die Angaben, die von Type erwartet werden.

Die Anwendung dieses Wortes erfolgt also üblicherweise in der Form
‚adr1 COUNT TYPE',
wobei als ‚adr1' die Adresse des COUNT-Bytes (Zeichenanzahl) übergeben wird. Der eigentliche String muß dem COUNT-Byte folgen.

KEY (→ c)

Dieser Befehl wartet auf eine beliebige Tastatureingabe und übergibt den entsprechenden ASCII-Code auf den Stack. Bei einigen Forth-Versionen ist die RETURN-Taste zur Quittierung erforderlich.
KEY . (RETURN)
A
65 OK
KEY KEY . . (RETURN)
A B
66 65 OK
KEY EMIT (RETURN)
D
D OK

Programmieren in Forth

Forth-Editor

Im letzten Teil haben wir das Definieren von neuen Befehlswörtern kennengelernt. Wir haben auch gelernt, daß in FORTH definieren gleichbedeutend mit Programmieren ist. Um jedoch größere Programme entwikkeln zu können, müssen wir diese abspeichern, laden und editieren können. Dies geschieht in FORTH – im Gegensatz zu Basic – mittels Texteditor. Der Texteditor ist leider in jeder FORTH-Version unterschiedlich zu handhaben. Wir wollen uns deshalb mit unserer Beschreibung auf den weitverbreiteten „FIG-STANDARD" beziehen.

Im Gegensatz zu vielen anderen Texteditoren wird in FORTH immer nur eine Bildschirmseite editiert. Man spricht dabei von einem „SCREEN". Diese SCREEN's werden mit einer Nummer versehen und in der Regel auf Diskette gespeichert. Je nach FORTH-Version können 20 bis über 100 solcher Screen's abgespeichert werden. Ein solcher Screen ist in 16 Zeilen zu je 64 Zeichen aufgeteilt. Da bei einigen Computern nur 40 Zeichen darstellbar sind, wird durch ein links/rechts-Scrollen ein größerer Bildschirm simuliert. Ein Beispiel dafür ist „64 FORTH" von HES. Die meisten FORTH-Versionen besitzen mehrere TEXTPUFFER. Diese erlauben das gleichzeitige Einladen mehrerer „SCREEN's".

Wollen wir einen Screen auf dem Bildschirm ausgeben lassen, so können wir dies mit dem Befehl: LIST. Das folgende Beispiel listet Screen Nummer 10:

10 LIST (RETURN)

Ein Screen könnte beispielsweise wie folgt aussehen:

```
 0  ( BEISPIEL EINES SCREENES;JUNI 84 )
 1
 2  : ROUND   SWAP
 3            500 + 1000 / +
 4            ;
 5  : SCALE   392 1000 */MOD ROUND ;
 6
 7  : M->T    SCALE . CR ;
 8
 9  : STERN   ." *" ;
10
11  ." DIES IST EIN SCREEN"
12
13
14
15
```

Die Zeilennummern dienen nur zur Orientierung und werden deshalb nicht von allen FORTH-Versionen ausgegeben. Die erste Zeile sollte immer einen Kommentar enthalten, um nicht den Überblick zu verlieren. Der Kommentar beginnt mit einem (Zeichen, gefolgt von einem Leerzeichen und geht bis zum Zeichen). In diesem Kommentar sollte das Datum und die Aufgabe des SCREEN's beschrieben werden. Selbstverständlich können auch in anderen Zeilen noch Kommentare notiert werden.

Wir haben nun gesehen, wie ein beschriebener SCREEN aussehen kann und wollen nun daran gehen, einen SCREEN zu beschreiben.

Zuerst suchen wir uns mit Hilfe eines Handbuches einen freien SCREEN aus. Mit . . . LIST können wir uns überzeugen, daß der SCREEN leer ist. Ist beispielsweise der SCREEN Nummer 10 unbeschrieben, so geben wir folgenden Befehl ein: 10 EDIT (RETURN).

Nun befinden wir uns im eigentlichen Editor. Dieser ist im FIG-STANDARD zeilenorientiert. Das bedeutet, daß der Text mit Hilfe mehrerer Kurzbefehle editiert bzw. erstellt wird.

Die folgende Aufstellung zeigt die Funktion der Standard-Befehle:

LINE n --» adr

Dieser Befehl übergibt dem Stack die Adresse der Zeilennummern. Ist der entsprechende Screen nicht im Puffer, so wird dieser von der DISK eingelesen.

-MOVE adr --» n

Die Textzeile ab Adresse adr wird in Zeile n kopiert.

H n --»

HOLD übernimmt die Textzeile n nach PAD und hält sie dort zur Ausgabe bereit.

E n --»

ERASE löscht Zeile Nummer n und füllt diese mit Leerzeichen (Blanks) aus.

S n --»

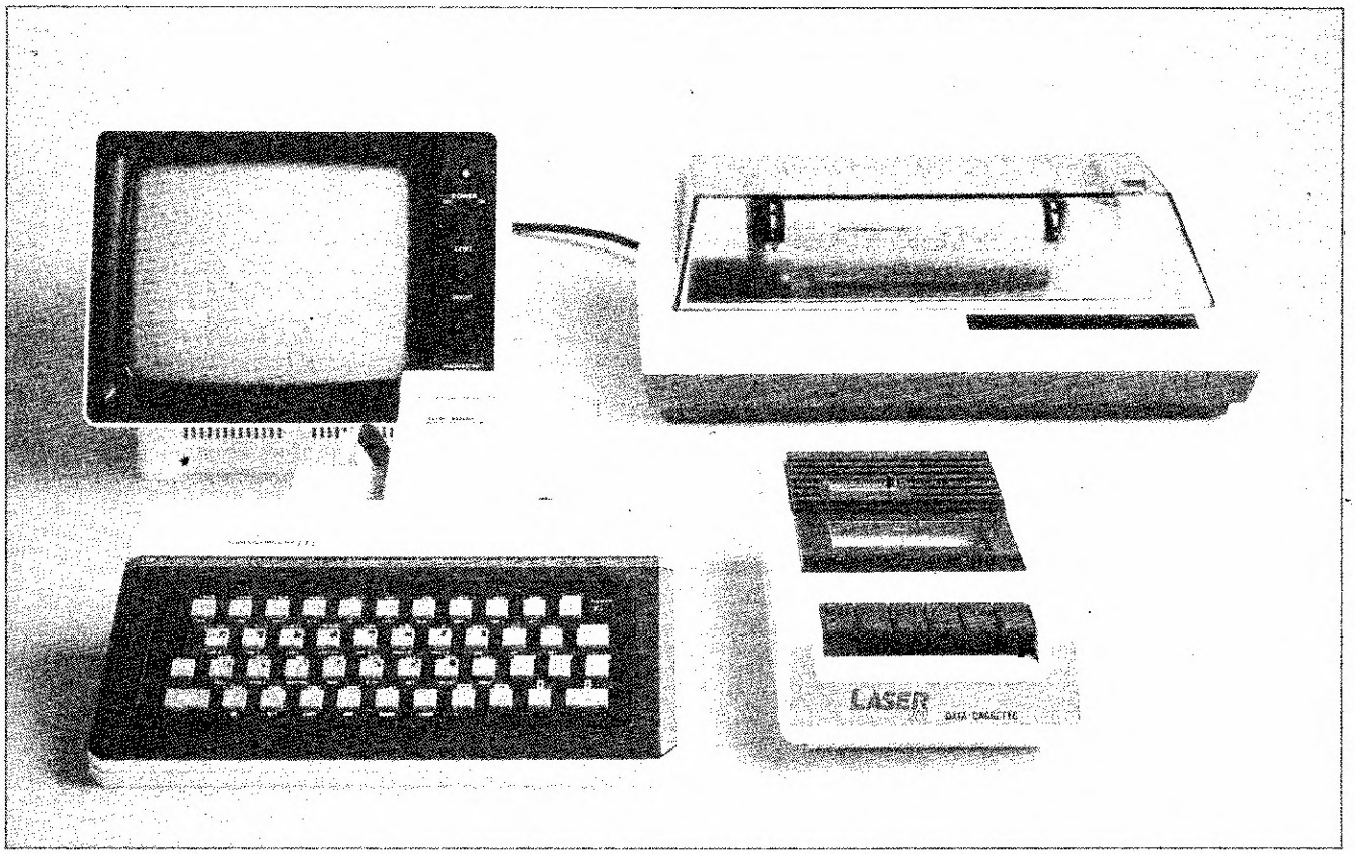
Dieser Befehl fügt vor Zeile n eine Leerzeile ein. Nachfolgende Zeilen werden um eine Zeile nach unten ver-

schoben. Die unterste Zeile geht verloren.
D n --»
Die Zeile n wird gelöscht. Dadurch wird der nachfolgende Text um eine Zeile angehoben.
T n --»
Zeile Nummer n wird auf dem Schirm ausgegeben.
L n --»
Der zur Zeit bearbeitete Screen wird ausgelistet.
R n --»
Dieser Befehl ist das Gegenstück zum H-Befehl. Die im PAD abgelegte Zeile wird in Zeile n ausgegeben.
P n --»
Der folgende Text wird in Zeile n übertragen. Der alte Zeileninhalt wird überschrieben.
I n --»
Der PAD wird ab Zeile n ausgegeben. Die nachfolgenden Zeilen werden um eine Zeile nach unten verschoben.
CLEAR n --»
Der Screen n wird mit Leerzeichen (Blanks) vollgeschrieben und somit gelöscht.
COPY n1 n2 --»
Dieser Befehl kopiert den gesamten Screen n1 nach Screen n2.
Die nachfolgend aufgestellten Befehle gehören zwar nicht zum Editor, sind jedoch zum Verwalten der Screen's erforderlich.
LIST n1 --»
Screen n1 wird auf dem Schirm ausgelistet.
FLUSH
Alle geänderten Screen's werden auf Diskette abgespeichert.
EDIT n1 --»
Screen n1 wird eingeladen und kann editiert werden.
LOAD n1 --»
Dies ist der wichtigste Befehl. Screen n1 wird in den Puffer eingeladen und dann Zeile für Zeile compiliert. Wird ein Wort mehrmals definiert, so wird dies mit einer Meldung angezeigt. Der Compilier-Vorgang wird in diesem Falle jedoch nicht unterbrochen. Um jedoch unnötigen Speicherverlust zu vermeiden, sollte man das Wörterbuch vor jedem neuen Compilier-Vorgang zurücksetzen. Wie wir wissen, wird dies mit dem Befehl FORGET . . . gemacht.

Sicher wird sich so mancher schon gefragt haben, wie er längere Programme schreiben soll, wenn doch immer nur ein Screen compiliert wird. Des Rätsels Lösung ist ganz einfach:
Mit Hilfe des --»-Befehls kann man beliebig viele Screen's miteinander verbinden und somit sehr lange Programme schreiben.
Der --»-Befehl wird in diesem Fall an das Ende des letzten Screen's gesetzt. Aufgerufen wird jeweils nur der erste Screen. Je nach Datenträger und Anzahl der Puffer dauert der Compiliervorgang bis zu einigen Minuten.
Einige Forth-Versionen besitzen auch die Möglichkeit, das Wörterbuch direkt abzuspeichern.
Im C64 FORTH von HES heißt der Befehl zum Abspeichern des Wörterbuches: DSAVE.

Das Einladen geschieht in diesem Fall mit DLOAD. Das D steht in beiden Fällen für Dictionary und ist gleichbedeutend mit Wörterbuch. Bei der Benutzung dieser beiden Befehle sollte man bedenken, daß nur der compilierte Code angespeichert wird und somit keine Änderungen mehr möglich sind.
Wir haben nun eine Menge über den FORTH-Texteditor erfahren und können somit auch längere Programme schreiben, ohne diese nach dem Ausschalten zu verlieren.
Wie schon erwähnt, ist es von größter Wichtigkeit, seine FORTH-Programme gut zu dokumentieren, da FORTH-Befehle allein sehr unübersichtlich wirken.
Zur Dokumentation ist übrigens an dieser Stelle zu sagen, daß Kommentare unbestreitbar den Vorzug haben, unverrückbar physikalisch an die kommentierte Definition geknüpft zu sein. Daher wird jede Änderung eines Programms auch mit der Aktualisierung der Kommentierung zwangsläufig verbunden sein. Dennoch sollte man die Kommentare auf der Diskette knapp halten. Genauer:
Die mit dem Code verknüpften Kommentare sollten knapp sein, da sonst die Compilierungszeit unnötig erhöht wird.
Daneben ist die Lesbarkeit von Texten innerhalb von Codes relativ schlecht und verschlechtert auch die Lesbarkeit des Codes.
Da die Diskettenlaufwerke von Heimcomputern auch keine unbegrenzte Speicherkapazität besitzen, sollte man sich überlegen, ob man die Dokumentation nicht wie bei Assembler-Programmen getrennt vom Programm vornimmt.
Wie wäre es denn z. B. mit Papier und Bleistift?

LASER®



LASER 110 – 4 KByte RAM. Dieser „persönliche Computer" eignet sich als Helfer beim Studium, am Arbeitsplatz oder bei Analysen und Statistiken. Mit seinen Peripheriegeräten erfüllt er alle Ansprüche an einen Home-Computer.

FORTH Kurs Teil 5

Bedingte Verzweigungen
Das Forth-Befehlswort „IF" eröffnet alle bedingten Verzweigungen in FORTH-Programmen. Dieses Befehlswort erwartet eine 16-Bit Zahl auf dem Stack und behandelt dieses als Flag. Der „IF"-Befehl arbeitet nur zusammen mit dem Befehlswort „ENDIF". „ELSE" ist ebenfalls möglich, jedoch nicht unbedingt nötig. Folgendes Beispiel zeigt den Zusammenhnag der einzelnen Befehle:

```
IF      (IF testet das Flag; falles es
        wahr (= 1) ist ...)

...     (werden die nebenstehenden
        Anweisungen ausgeführt

...     (ansonsten)

ENDIF(fortgefahren)

IF
...     (falls Flag = 1, werden neben-
        stehende Worte)

...     (bis ELSE ausgeführt)

ELSE
...     (andernfalls erfolgt die Ausfüh-
        rung der Worte)
...     (zwischen ELSE und ENDIF)

ENDIF
```

Bei der Verwendung von Verzweigungen sollte man immer bedenken, daß diese nicht interprevativ verwendet werden dürfen, also nur innerhalb einer Definition. Probieren wir doch einmal die folgende, einfache Definition:

```
:DIVISION(Division mit Kommentar,
          falls Rest nicht Null) (Para-
          meter wie bei / – Befehl)

/MOD      (Division mit Rest)

SWAP      (Bringe Rest zum TOS und
          teste ihn als Flag)

IF"DIVISION      geht      nicht
auf"

ENDIF
;         (falls Rest = „wahr", wird
          der Text gedruckt)
```

Das nun definierte Wort „DIVISION" erwartet also zwei Zahlen auf dem Stack, dividiert diese und beläßt den Quotienten auf dem TOS. Die Syntax des neuen Befehls entspricht genau dem Original "/" Befehl. Im Gegensatz zu diesem, wird jedoch bei Entstehung eines Restes die Meldung „DIVISION GEHT NICHT AUF", ausgegeben.

Wir wollen nun ein Beispiel zeigen, welches den „ELSE"-Befehl vorführt. Es handelt sich hier um einen Vergleichs-Befehl, welcher angibt, ob die erste Zahl kleiner oder größer als die zweite Zahl ist.

```
:VERGLEICHE (Vergleich zweier
            Zahlen)

<           ."(FLAG ermitteln)

IF          ."KLEINER"

ELSE        ."GROESSER"

ENDIF
;
```

Beispiel:
33 100 VERGLEICHE (RETURN)
KLEINER OK
100 33 VERGLEICHE (RETURN)
GROESSER OK

Zum Vergleich dasselbe noch einmal in der Programmiersprache Basic:

```
10 INPUT „ZAHL1";Z1

20 INPUT „ZAHL2";Z2

30 IF Z1<Z2 THEN PRINT „KLEI-
NER"

40 IF Z1>Z2 THEN PRINT „GROES-
SER"

50 END
```

Da im Standard-Basic keine „ELSE"-Befehl zugelassen ist, muß man sich mit zwei „IF" Anweisungen behelfen, wenn man nicht zu dem UNSAUBEREN „GOTO"-Befehl greifen will. Dies geht selbstverständlich auf Kosten der Zeit.
Im übrigen sei an dieser Stelle darauf hingewiesen, daß einige FORTH-Versionen statt des Befehles „ENDIF" den Befehl „THEN" verwenden. Dies muß in dem entsprechenden Manual nachgelesen werden.
Wir haben nun einiges über die „IF"-Anweisung gelernt und wollen uns nun die eigentlichen Vergleichsbefehle näher ansehen:

< (n1 n2 n3 --> n1 f)
Flag wird 1 wenn die Zahl n2 kleiner als n3 ist.

> (n1 n2 n3 --> n1 f)
Flag wird 1 wenn die Zahl n2 größer als n3 ist.

= (n1 n2 n3 --> n1 f)
Flag wird 1 wenn beide Zahlen gleich sind

0< (n1 n2 n3 --> n1 n2 f)
Flag wird 1 wenn n3 eine negative Zahl ist

0= (n1 n2 n3 --> n1 n2 n3)
Flag wird 1 wenn die Zahl n3 gleich null ist.

Als Ergebnis des Vergleiches wird immer eine Boolsche Zahl abgelegt. Diese wird auch als Flag bezeichnet und gibt an, ob die Aussage richtig oder falsch ist. Ist diese richtig, so wird eine 1, ansonsten eine 0 auf den Stack gelegt.

Beispiele:

```
23 30 < . <RETURN> 1 OK
30 23 < . <RETURN> 0 OK
30 44 = . <RETURN> 0 OK
-2  0 < . <RETURN> 1 OK
```

Bei dem Vergleich zweier Zahlen sollte man bedenken, daß diese vom Vergleichsbefehl überschrieben werden. Will man diese jedoch für weitere Berechnungen erhalten, so kann man dies mit dem Befehl OVER erreichen:
23 30 OVER OVER < . . . <RETURN> 1 30 23

Programmschleife DO...LOOP

Ein wichtiger Bestandteil aller Programmiersprachen sind Schleifen. Unter Schleife versteht man einen oder mehrere Befehle, die durch zwei Kennzeichnungen eingeklammert sind. Diese eingeklammerten Befehle werden beliebig oft durchlaufen. Ist eine bestimmte Anzahl durchlaufen, wird das Programm fortgesetzt. In FORTH gibt es genau wie in Pascal keinen „GOTO" Befehl, deshalb ist der Programmierer auf die Schleife angewiesen.

Der Schleifenbefehl hat in FORTH folgende Syntax:

DO ... LOOP (n1 n2 n3 --> n1)
Die Schleife wird von n3 bis n2 in Einerschritten durchlaufen. Die Zahl n3 muß in diesem Fall kleiner als n2 sein.

Die oberste Zahl auf dem Stack wird immer als Anfangswert und die darunterliegende Zahl als Endwert verwendet. Während der Schleife wird der Zahlenparameter immer um eins erhöht, deshalb muß der Endwert immer größer als der Anfangswert sein. Ist der Zahlenparameter größer oder gleich dem Endwert, so wird die Schleife verlassen. Da diese Kontrolle jedoch immer erst am Ende einer Schleife durchgeführt wird, ist die Schleife bereits einmal durchlaufen.
Will man während eines Schleifendurchlaufs den Zahlenparameter auslesen, kann man dies mit Hilfe des Wortes „I". Dieses Wort legt den Zahlenparameter auf den Stack.

Wie der „IF" Befehl kann auch die DO-LOOP-Schleife nicht interpretativ ausgeführt werden und muß deshalb innerhalb einer Definition stehen.
Das folgende Beispiel gibt die Zahlen 0–12 auf dem Schirm aus:

: ZAHLEN 12 0 DO I . LOOP ; <RETURN> OK

ZAHLEN <RETURN> 0 1 2 3 4 5 6 7 8 9 10 11 12 OK

Das gleiche in BASIC:
FOR I = 0 TO 12 : PRINT I, :NEXT 1

Ist der Endwert kleiner als der Anfangswert, so wird die Schleife nur einmal durchlaufen:

: ZAHLEN 0 12 DO I . LOOP ; <RETURN> OK
ZAHLEN <RETURN> 12 OK

Das nächste Beispiel zeigt zwei DO-LOOP-Schleifen die ineinander verschachtelt sind. Der Aufruf der neuen Definition gibt die Zahlen 0 bis 9 in einer Dreieckmatrix aus.

: SCHLEIFE1 0 DO I . LOOP ; <RETURN> OK

: SCHLEIFE2 0 DO CR 1 - DUP SCHLEIFE1 LOOP ; <RETURN> OK
11 10 SCHLEIFE2 <RETURN> OK

0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6
0 1 2 3 4 5
0 1 2 3 4
0 1 2 3
0 1 2
0 1
0 OK

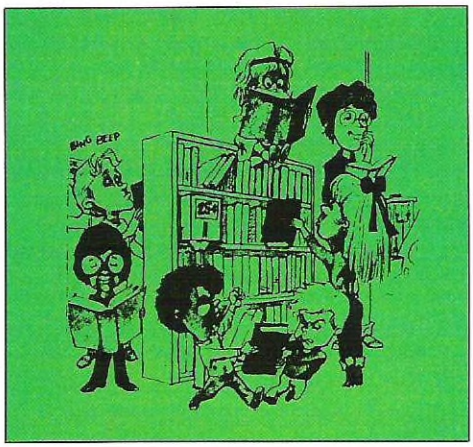
In vielen Programmen werden DO-LOOP-Schleifen zur Verzögerung von Programmabschnitten verwendet. Durch den folgenden Vergleich wird ersichtlich, wie schnell die Programmiersprache FORTH arbeitet:

Eine Leerschleife in FORTH:
: PAUSE 10000 0 DO LOOP ;

Die Ausführungszeit dieser Schleife beträgt im Durchschnitt 4 Sekunden. Je nach FORTH Version und Microprozessor kann diese Zeit etwas länger oder auch kürzer werden.

Die gleiche Schleife in BASIC:
10 FOR I = 0 TO 10000 : NEXT 1

In Basic ist die durchschnittliche Ausführungszeit etwa 4 bis 5 mal so lang wie in FORTH. Wie zu erkennen ist, eignet sich somit FORTH auch für Programme, in denen es auf genaue „Pause-Zeiten" ankommt. (Z. B. Eprom-Programmierung oder sonstige Ein- bzw. Ausgaben).
So mancher wird sich inzwischen fragen, ob man das Zahlenparameter nicht auch um 2, oder einer anderen Zahl erhöhen kann. Dies geschieht in Basic durch den Befehl „STEP".
Dies ist auch in FORTH möglich, indem man statt des „LOOP"-Befehls ein „+LOOP" einsetzt. In diesem Fall wird die Zahl, welche vor dem „+LOOP" auf dem Stack liegt, als Increment benutzt. Ist diese Zahl im negativen Bereich, so wird die Schleife bei jedem Durchlauf erniedrigt. In diesem Fall muß der Endwert kleiner als der Anfangswert sein, da sonst nur ein Durchlauf erfolgt.



Die Syntax: +LOOP (n1 n2 n3 --> n1 n2)

Einige Beispiele:

: ZAHLEN 10 0 DO I . 2 +LOOP ; <RETURN> OK

ZAHLEN <RETURN> 0 2 4 6 8 OK
In BASIC:

10 FORI = 0 TO 10 STEP 2 : PRINT I, : NEXT I

Umgekehrte Schleife:

: ZAHLEN 0 10 DO I . -2 +LOOP ; <RETURN> OK

ZAHLEN <RETURN> 10 8 6 4 2 OK

In Basic:

10 FORI = 10 TO 2 STEP -2 : PRINT I, : NEXT I

Bei Verwendung des „+LOOP" sollte man daran denken, daß vor dessen Ausführung eine 0 zur Endlosschleife führt.

BEGIN...UNTIL-Schleife

Da in FORTH kein Sprungbefehl vorhanden ist, müssen einige spezielle Schleifen diese ersetzen. Eine davon ist die BEGIN-UNTIL-Schleife.
Diese Schleife wird so lange durchlaufen, bis vor dem Befehl „UNTIL" ein „WAHR-Flag" gesetzt wird. Eine 0 führt zu einem neuen Durchlauf. Diese BEGIN-UNTIL-Struktur kann also sehr leicht den „GOTO"-Befehl ersetzen und führt somit zu einem übersichtlicheren Programm.
Die Struktur der Schleife sieht wie folgt aus:

```
BEGIN
  ...        (Befehle innerhalb der
             Schleife)
  ...
  ...
  ...        (Testteil der Schleife z. B.
             Vergleich)
             (Flag wird auf Stack gelegt)
UNTIL        (Ist Flag=0 erfolgt Sprung
             nach Begin)
  ...        (Flag ist ungleich 0; Pro-
             gramm wird fortgesetzt)
  ...
```

Das Beispiel mit den Zahlen:

```
: ZAHLEN
    0
    BEGIN
      1+ (Zahl um 1 erhöhen)
      DUP .        (Zahl ausgeben)
      DUP 10 =     (Ist 10 erreicht?)
    UNTIL
;
```

ZAHLEN <RETURN> 1 2 3 4 5 6 7 8 9 10 OK

In dem obigen Beispiel wird die Schleife so lange wiederholt, bis die auf dem Stack liegende Zahl den Wert 10 erreicht hat.

Eine weitere Anwendung wäre beispielsweise das Abfragen der Tastatur:

```
: INKEY (Auf <RETURN> wird ge-
wartet)
    BEGIN
      KEY
      13 = (Ist <RETURN> ge-
drückt)
    UNTIL
  . "RETURN WURDE GE-
DRUECKT"
;
```

INKEY <RETURN>

<RETURN> RETURN WURDE GEDRUECKT OK

Zum Schluß das Beispiel noch einmal in Basic:

```
10 GET E$: IF E$="" THEN 10
20 IF ASC(E$(<> 13 THEN 10
30 PRINT „RETURN WURDE GE-
DRUECKT"
```

Forth-Kurs

Teil 6

Die Endlosschleife Begin ... Again

Diese Programmschleife kennt keine Aussprungbedingung, ist also endlos. Die Hauptanwendung dieses Befehlswortes liegt im Aufruf bzw. in der unbegrenzt häufigen Abarbeitung eines (periodisch zu durchlaufenden) Hauptprogrammes. Die Syntax dieser Schleife entspricht weitgehend der BEGIN-UNTIL-Schleife.

Beispiel:

```
: TEST
      BEGIN
        . "COMPUTRONIC" CR (diese Anweisung wird endlos ausgeführt)
      AGAIN
;
```

Zu bemerken ist noch, daß es sich bei dieser Schleifenart um die schnellste FORTH-Schleife handelt.

BEGIN-WHILE-Schleife

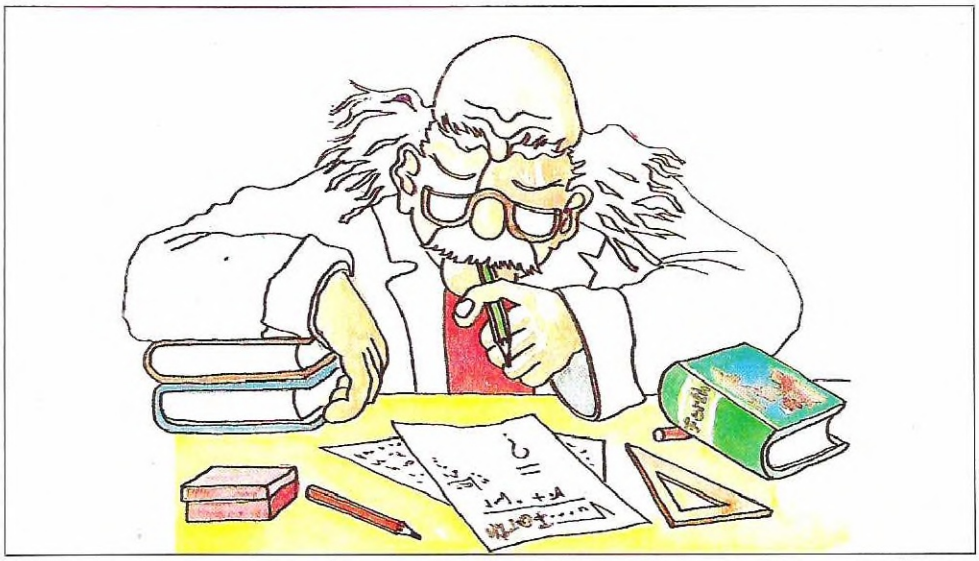
Auch diese Schleifenstruktur wird beliebig lange durchlaufen. Im Gegensatz zur BEGIN-UNTIL-Schleife erfolgt die Abfrage einer Aussprungbedingung (Flag) am Anfang des Schleifenkernes. Ist die Bedingung am Schleifenbegin nicht erfüllt, so wird die Schleife nicht ein einziges Mal durchlaufen. Das folgende Beispiel zeigt die Syntax dieser Schleife:

```
BEGIN
...        (Bedingung zur Erzeugung des Flags)
WHILE      (solange das Flag = 1 ist,
...        wird der folgende Teil durchlaufen)
REPEAT     (Rücksprung nach BEGIN)
```

In dem folgenden Beispiel wird die Tastatur abgefragt. Ist die Taste „C" gedrückt, so wird der Text „TASTE C IST GEDRÜCKT" ausgegeben. Die Schleife wird erst verlassen wenn eine andere Taste gedrückt wird.

```
: TASTE
    BEGIN
      KEY 67 - 0 = (Ist Taste „C" gedrückt?)
        WHILE
        . "TASTE C IST GEDRÜCKT" CR
      REPEAT (Springe zurück)
;
```

Wir haben nun alle verfügbaren FORTH-Schleifen kennengelernt. Es gibt in FORTH auch die Möglichkeit noch weitere Arten von Schleifen zu definieren, doch dies ist dem etwas erfahreren Programmierern vorbehalten. Wer sich genauer damit beschäftigen möchte, muß auf andere Literatur zurückgreifen, da die Beschreibung dieser Definitionen den Rahmen dieses Kurses sprengen würde.



FORTH-Programmbeispiele:

Berechnung des größten gemeinsamen Teilers zweier Zahlen:

```
: TEILER (Größter gemeinsamer Teiler errechnen)
    BEGIN
      SWAP OVER MOD DUP
      0 =
    UNTIL
    DROP
;
27  21 TEILER ‹RETURN› 3 OK
 8   6 TEILER ‹RETURN› 2 OK
```

Die Berechnung wurde mit Hilfe des Euklidschen Algorithmus verwendet. Zuerst bestimmt man den Rest ›R‹ zweier Zahlen, ›A‹ und ›B‹, durch (A MOD B). Ist R=0, dann ist B der gesuchte Teiler. Ist R‹›0, so wird A=B und B=R. Danach wir die obige Division wieder ausgeführt. Solange, bis die Bedingung R=0 erfüllt ist.

Variablen und Konstanten in Forth

Obwohl die meisten Zahlenwerte und Rechenoperationen über den Stack geführt werden, kann man trotzdem auch Konstante und Varibalen definieren.

In Basic wird eine Konstante wie folgt definiert:

```
10 LET Y = 10
```

Der Konstanten Y wird der Zahlenwert 10 zugeordnet. In FORTH würde das gleiche so geschrieben:

```
10 CONSTANT Y ‹RETURN› OK
```

Durch diese Befehlsfolge wird im Wörterbuch eine Konstante „Y" eingetragen, die den Zahlenwert 10 enthält. Sobald der Variablennamen aufgerufen wird, erhält der Stack den entsprechenden Zahlenwert.

Beispiele:

```
12 Y + .  ‹RETURN›  22 OK
 5 Y ★ .  ‹RETURN›  50 OK
 Y Y ★ . ‹RETRUN› 100 OK
```

Das ganze kann auch in einer Definition stehen:

```
: Test
    Y ★ .
;

7 TEST ‹RETURN› 70 OK
```

Wie wir wissen, können wir statt der Konstanten auch einfach folgendes schreiben:

```
: Y 10 ;
```

Dies hat zwar die gleiche Wirkung wie die Definition einer Konstanten, ist jedoch in der Ausführungszeit wesentlich länger.

Im Gegensatz zu den meisten BASIC-Versionen können die Namen der Variablen, Konstanten und Befehlen in den meisten FORTH-Versionen bis zu 31 Zeichen lang sein. Es ist so möglich, den Variablen einen sinnvollen Namen zu geben, um somit das Programm übersichtlicher zu machen.

Ähnlich wie die Konstanten werden in FORTH auch die Variablen definiert:

10 VARIABLE TAG

In diesem Beispiel wurde der Variablen „TAG" der Wert 10 zugewiesen. Der Unterschied zwischen Variablen und Konstanten besteht darin, daß die Konstante nach der Definition ihren Wert immer behält. Dieser wird beim Aufruf der Konstanten auf den Stack gelegt. „Beim Aufruf einer Variablen wird dagegen nicht der Wert der Variablen, sondern die Adresse dieser Variable auf den Stack gelegt. Zum Ändern bzw. Auslesen des Wertes einer Variablen, benötigen wir noch zwei weitere Befehle, nämlich den Befehl :Wert holen (Load) ; und den Befehl :Wert wegspeichern (STORE).

Für den STORE-Befehl wird in Forth das „!" Zeichen verwendet. Dieser Befehl entspricht weitgehend dem bekannten POKE aus dem BASIC. Jedoch wird hier keine 8-Bit-Zahl, sondern eine 16-Bit-Zahl abgespeichert. Die Syntax des STORE-Befehl:

! (n1 n2 adr --› n1)

Für den LOAD-Befehl wird in Forth das " " Zeichen verwendet. Dieser Befehl ist mit dem Peek-Befehl im Basic zu vergleichen. Auch hier wird eine 16-Bit-Zahl ausgelesen:

Die Syntax des LOAD-Befehl:

@(n1 n2 adr --› n1)

Die eben vorgestellten Befehle LOAD und STORE arbeiten nur mit 16-Bit-Zahlen. Will man dagegen nur ein einziges Byte, also eine Dezimalzahl zwischen 0 und 255 abspeichern, so verwendet man das Wort C! „Mit Ce wird ein Byte auf den Stapel geholt. Dabei werden die oberen 8 Bits zu Null gesetzt.

Wenn wir nur den Variablennamen, gefolgt von einem Dezimalpunkt (Print) eingeben, so erhalten wir die Adresse der Variablen. Diese kann je nach FORTH-Version und Anzahl der Definitionen verschieden sein.

TAG . ‹RETURN› 4104 OK

Möchte man den Wert der Variablen TAG auslesen, so geschied dies wie folgt:

TAG @ . ‹RETURN› 10 OK

Das Ändern einer Variablen geht ebenso einfach:

13 TAG ! ‹RETURN› OK
(Der Wert 13 wird gespeichert)

TAG @ . ‹RETURN› 13 OK
(Der Wert wird ausgelesen)

Auf ähnliche Weise kann man auch den Wert einer schon vereinbarten Konstanten ändern. Die Adresse einer Konstanten erhält man mit dem Befehlswort "'" (Einzelapostroph).
Die Syntax:
' (Name) (n1 n2 --› n1 n2 adr)
Die Adresse der Konstanten (Name) wird auf den Stack gebracht.

Beispiele:

20 CONSTANT TRONIC ‹RETURN› OK
(der Konstanten TRONIC wird der Wert 20 zugewiesen)

TRONIC . ‹RETURN› 20 OK
(der Wert wird ausgelesen)

30 ' TRONIC ! ‹RETURN› OK
(der Wert 30 wird in die Konstante geschrieben)

TRONIC . ‹RETURN› 30 OK
(nochmal überprüfen)

FORTH-Zahlensystem

Bei der Initialisierung von Forth, wird immer das Rechnen zur Basis 10, also die gewohnte Dezimalrechnung angenommen. Forth kann jedoch auch mit allen anderen Zahlensystemen rechnen. Will man z. B. mit Hexadezimalzahlen arbeiten, so kann mit dem Befehl „HEX" auf diesen Modus umgeschaltet werden. Danach erfolgen alle Ein- und Ausgaben in hexadezimaler Schreibweise.

HEX (n1 n2 n3 --› n1 n2 n3)

Möchte man wieder in den Dezimal-Modus zurück, so geschieht dies mit dem Befehl "DECIMAL".

DECIMAL (n1 n2 n3 --› n1 n2 n3)

Beispiel:

DECIMAL 90 HEX . ‹RETURN› 5A OK
Die Dezimalzahl 90 entspricht der Hexadezimalzahl 5A.

HEX 5A DECIMAL . ‹RETURN› 100 OK
Das umgekehrte Beispiel.

Möchte man ein anderes Zahlensystem verwenden, muß man in die System-Variable BASE den entsprechenden Wert speichern. Durch den Befehl HEX wird eine 16 und durch DECIMAL eine 10 in die Variable geschrieben.

Möchten wir beispielsweise das Binärsystem programmieren, so müssen wir nur die Zahl 2 (Zustand 0 und 1) in die Variable BASE bringen. Den Umschalt-Befehl nennen wir BIN.

: BIN 2 BASE ! ;

Mit dem Befehl BIN können wir nun jederzeit auf Binär-Modus umschalten.

DECIMAL 255 BIN .
‹RETURN› 11111111 OK
DECIMAL 16 BIN .
‹RETURN› 10000 OK
BIN 01010101 DECIMAL .
‹RETURN› 85 OK
BIN 00010000 DECIMAL .
‹RETURN› 16 OK

Zum Abschluß noch ein kleines FORTH-Programm:

Berechnung der Primzahlen zwischen zwei Grenzen.

```
4 VARIABLE ZEILE
: TEST      MOD 0=;
: DRUCKE    DUP 4 .R ZEILE e DUP 0=
            IF CR DROP 4
            ELSE 1 -
            THEN
            DROP ;
: PTEST     DUP 2 / 2 DO DUP I TEST
            IF 0 LEAVE
            THEN LOOP DUP IF
            DRUCKE
            ELSE DROP
            THEN
            DROP ;
: PRIM      (ende anf --› )
CR 4        ZEILE !
DO I        PTEST LOOP CR ;
```

200 1 PRIM ‹RETURN›

1	2	5	7	11
13	17	19	23	29
31	37	41	43	47
53	59	61	67	71
73	79	83	89	97
101	103	107	109	113
127	131	137	139	149
151	157	163	167	173
179	181	191	193	197
199	OK			

Forth-Kurs

Auf vielfachen Wunsch hier unser kompletter Kurs

Vorstellung der Programmiersprache

Wer schon in Basic programmiert hat wird wissen, daß es hin und wieder Anwendungen gibt, die recht schwierig zu programmieren sind. Für solche Fälle hat man bei großen Rechnern die Möglichkeit, auf andere Sprachen zurückzugreifen. Es gibt *FORTRAN* für mathematische Problemlösungen, *COBOL* für kommerzielle Zwecke, *ASSEMBLER* für zeitkritische Aufgaben, *BASIC* für die allgemeinen Problemlösungen auf Microcomputern.

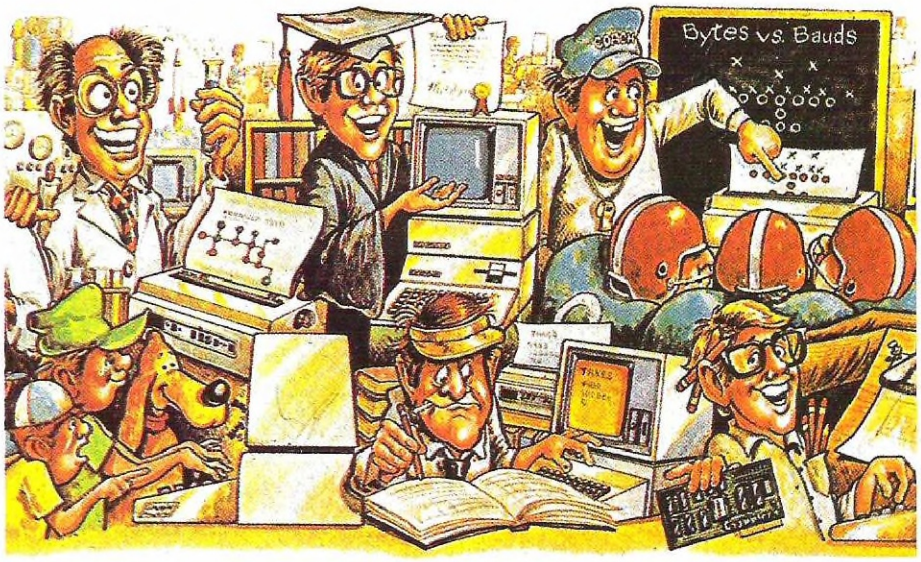
FORTH gehört zu der jüngsten Generation von Programmiersprachen. Versucht wurde, alle Vorteile der renommierten Programmiersprachen in *FORTH* zu implementieren, ohne aber deren „Fehler" mit zu übernehmen.

FORTH hat schon in seiner Struktur einige markante Vorteile, die besonders bei den Microcomputern von Vorteil sind:

- *FORTH* ist auf fast allen Microcomputern implementierbar und benötigt nur wenig Speicherplatz.
- *FORTH*-Programme sind wesentlich kürzer als *BASIC*- und *ASSEMBLER*-Programme.
- *FORTH* ist in der Regel 10mal schneller als *BASIC*.
- *FORTH* kann auch mit Kassettenrecorder betrieben werden. Vorteilhaft ist aber ein Floppy-Laufwerk.
- *FORTH* ist problemlos erweiterbar.
- *FORTH*-„Hochsprache" und *FORTH*-„Assembler" lassen sich beliebig mischen.
- *FORTH* erlaubt eine strukturierte Programmierung.

FORTH ist ursprünglich für die Programmierung von Steuerungsaufgaben entwickelt worden. Da die Sprache leicht erweiterbar ist, kann sie für fast alle Anwendungen verwandt werden. Durch das Implementieren einiger Grafik-Befehle wird aus der Prozeßsprache schnell eine geeignete Sprache für Spiele und Grafik-Anwendungen. Jeder kann *FORTH* um die Befehle erweitern, die er für seine Anwendungen gerade benötigt. Trotz zahlreicher Erweiterungen sind *FORTH*-Programme sehr leicht auf einen anderen Rechner zu übertragen. Fehlende Worte (Befehle) können einfach implementiert werden.

Wir hoffen, daß es uns gelungen ist, einige Vorteile von *FORTH* darzustellen. Auf den folgenden Seiten werden Sie schrittweise mit den Eigenheiten der Sprache vertraut gemacht. Es ist schade, daß bisher kaum eine Zeitschrift näher auf *FORTH* eingegangen ist. Obwohl es sicher eine der leistungsfähigsten Sprachen für Microcomputer ist und immer mehr Freunde findet.



Einleitung

FORTH ist ein selbständiges Sprachsystem, das sowohl Compiler als auch Interpreter ist und sein eigenes Betriebssystem beinhaltet. Das Besondere an *FORTH* ist, daß mit dem „Stapel" gearbeitet wird. Wer schon einmal in Assembler programmiert hat, kennt die Arbeitsweise des Stapels. Der Stapel arbeitet nach dem *LIFO-System*. Dies bedeutet, etwas wird abgelegt und kann später wieder heruntergenommen werden. *FORTH* besitzt zwei Stapel. Den Daten- und den Return-Stapel. Auf die Funktion wird später noch eingegangen.

Die Verwendung eines Stapels wäre eigentlich nichts Neues. Jeder HP-Taschenrechner verwendet einen Stapel zum Abspeichern von Zahlen. Der Unterschied zu den HP-Rechnern ist hier das Wörterbuch, welches auch als Stapel ausgelegt ist.

Die Sprache besteht aus Worten, die nach ihrer Definition auf dem Definitionsstapel, dem Wörterbuch, abgelegt werden. Bei der Erstellung eines Wortes können schon vorherige Definitionen verwendet werden. Ein *Forth-Programm* besteht also in der Regel aus zahlreichen Definitionen, die sich gegenseitig aufrufen.

Um Wörter wieder löschen zu können, benutzt man den Befehl >FORGET<. Der Aufruf >FORGET TEST< löscht alle Worte, beginnend bei >TEST< bis zum obersten Eintrag in den Stapel. Das Löschen eines einzigen Wortes im Stapel ist nicht möglich und wäre auch nicht sinnvoll. Denn spätere Definitionen könnten nicht mehr benutzt werden. Es gibt aber auch die Möglichkeit, ein Wort mehrmals zu definieren. In diesem Fall wird immer das zuletzt definierte Wort ausgeführt.

Berechnungen erfolgen in *FORTH* über den Datenstapel, den wir auch als STACK bezeichnen können. Da alle Operanden über den Stack geleitet werden, ergibt sich die etwas ungewohnte Rechenschreibweise >UPN< (Umgekehrte Polnische Notation). Auch die schon genannten HP-Taschenrechner verwenden diese Art der Eingabe. Um beispielsweise die Zahlen 5 und 7 zu addieren, muß folgendes eingegeben werden:

in FORTH: 5 7 + *in BASIC:* 5 + 7

Nach der Eingabe dieses Beispiels werden die beiden Summanden vom Stack geholt und addiert. Das Ergebnis wird wieder auf den Stack geschrieben. Man kann dies leicht überprüfen, indem man sich das Ergebnis durch den Befehl >"."< ausgeben läßt. Der Befehl >"."< holt die oberste Zahl

weiter Seite 31

Fortsetzung von Seite 4
vom Stack und gibt sie auf dem Schirm aus. Ein nochmaliges Aufrufen des Befehles >"."< bewirkt eine Fehlermeldung, da der Stack keine Zahl mehr beinhaltet. Wichtig ist, daß die einzelnen Worte und Zahlen immer durch mindestens ein Leerzeichen getrennt werden, da der Computer das Wort sonst nicht erkennen kann. Nach erfolgreicher Ausführung eines Wortes meldet sich der Computer mit „OK", dies entspricht in BASIC der „READY"-Meldung.
FORTH ist eine sehr nachsichtige Sprache. Alles was eingegeben und nicht direkt als falsch erkannt wird, wird angenommen und eventuell sogar ins Wörterbuch eingetragen. Man darf also nicht blindlings drauflosprogrammieren, sondern man sollte sich überlegen, was und wie man es programmiert.
Bei der Beschreibung des Wörterbuches und der Operationen mit dem Stapelspeicher (Stack) werden Bezeichnungen verwendet, wie sie im ⟨Installation Manual⟩ für *FORTH* festgelegt sind. Diese Bezeichnungen sollten Sie sich einprägen, da sie immer wieder verwendet werden.
Diese Bezeichnungen sind:

adr	Speicheradresse
b	8 bit Byte (die oberen 8 Bit sind null)
c	7 bit ASCII-Zeichen
d	32 bit doppelt lange Festkommazahl mit Vorzeichen
f	Boolsche Variable Ø = „false" (falsch)
n	16 bit-Zahl mit Vorzeichen
u	16 bit-Zahl ohne Vorzeichen
⟨RETURN⟩	RETURN-Taste

Bei der Beschreibung eines Wortes wird künftig folgendes Schema angewendet:
WORT ⟨RETURN⟩ Anzeige des Computers (Stapel vorher --> Stapel nachher)
KOMMENTAR:
Beispiel:
= ⟨RETURN⟩ OK (n1 n2 n3 --) n1 n2)
>n3< wird auf dem Bildschirm ausgegeben
Wie Sie schon wissen, ist >"."< eine *Forth-Anweisung,* die die oberste Zahl des Stapels auf dem Schirm ausgibt. Dies ist also die Print-Anweisung. Vor der Ausführung des Befehles befinden sich 3 Zahlen, nämlich n1, n2, n3 im Stapelspeicher. Davon ist die am weitesten rechts stehende Zahl im TOS (Top of Stack). In diesem Fall also ›n3‹. Nach der Ausführung des Befehles sind nur noch zwei Zahlen (n1, n2) im Stapel, wobei nun ›n2‹ an oberster Stelle steht.
Um noch einmal die Funktion des Stapels zu verdeutlichen, geben Sie bitte folgendes Beispiel ein:
1 2 3 4 5 6 ⟨RETURN⟩ OK (empty --) n1 n2 n3 n4 n5 n6)
Die Zahlen 1 bis 6 wurden nacheinander auf den Stapel gelegt. Um sie wieder abzurufen, geben Sie folgendes ein:
= ⟨RETURN⟩ 6 OK (n1 n2 n3 n4 n5 n6 --) n1 n2 n3 n4 n5)
= ⟨RETURN⟩ 5 OK (n1 n2 n3 n4 n5 --) n1 n2 n3 n4)
= ⟨RETURN⟩ 4 OK (n1 n2 n3 n4 --) n1 n2 n3)
= ⟨RETURN⟩ 3 OK (n1 n2 n3 --) n1 n2)
= ⟨RETURN⟩ 2 OK (n1 n2 --) n1)
= ⟨RETURN⟩ 1 OK (n1 --) empty)
Wie aus dem Beispiel zu erkennen ist, werden die Zahlen in der umgekehrten Reihenfolge ausgegeben. Natürlich ist es gleich, ob man den >"."< Befehl einzeln mit ⟨RETURN⟩ quittiert, oder nur durch ein Leerzeichen trennt.

Arithmetik-Operationen

Die 4 Grundrechenarten sind in *FORTH* und BASIC durch die gleichen Bezeichnungen gekennzeichnet.
+ [n^1 n^2 - -> Summe] Addition
- [n^1 n^2 - -> Differenz] Substraktion
* [n^1 n^2 - -> Produkt] Multiplikation
/ [n^1 n^2 - -> Quotient]Division
Wie bei allen *FORTH-Wörtern,* werden auch hier die Operanten auf dem Stack erwartet, und nach der Ausführung durch das Ergebnis überschrieben.
Einige Beispiele verdeutlichen die Arbeitsweise der Befehlswörter:
123⟨RETURN⟩ OK
[empty - -> n^1 n^2]
+ ⟨RETURN⟩ OK
[n^1 n^2 - -> Summe]
. ⟨RETURN⟩ 15 OK
[Summe - -> empty]
Eine andere Schreibweise:
123 + . ⟨RETURN⟩ 15 OK
43213 + . ⟨RETURN⟩ 256 OK
1000344 + . ⟨RETURN⟩ 1344 OK
500100 50 + + . ⟨RETURN⟩ 650 OK
Die Aufgaben 14–4, 12 * 3 und 2000/50 werden in *FORTH* wie folgt gelöst:
144 - . ⟨RETURN⟩ 10 OK
123 * . ⟨RETURN⟩ 36 OK
200050 / . ⟨RETURN⟩ 40 OK
Alle oben aufgeführten Wörter, sind INTEGER-Operationen. Dies bedeutet, daß die Operanten sowie das Ergebnis intern, 16 Bit belegen. Für die Anwendung ergibt sich somit folgender Zahlenbereich:
-32768 <- - - - - - - - - - - 0 - - - - - - - - - - -> 32767
Dieser Zahlenbereich darf nicht überschritten werden, da sich sonst das Ergebnis entsprechend verfälscht. Gleitkommazahlen sind in *FORTH* unzulässig.
Folgende Beispiele zeigen die Grenzen der Integer-Arithmatik:

10000 4 * .	⟨RETURN⟩ -25536 OK Richtig wäre 10000
32767 1 + .	⟨RETURN⟩ -32768 OK Richtig wäre 32768
5 2 / .	⟨RETURN⟩ 2 OK Richtig wäre 2.5
11 2 / .	⟨RETURN⟩ 5 OK Richtig wäre 5.5

Einige Basic-Freunde werden sich jetzt fragen: »Was soll ich mit einer Programmiersprache die nicht einmal richtig 5 durch 2 teilen kann?« Diese Frage ist sicher nicht ganz unberechtigt, jedoch sollte man bedenken, daß in sehr vielen Spiel-, Grafik-, Steuerungs- und Rechenprogrammen eine Integer-Arithmetik völlig ausreicht. Eine Gleitkommazahl können wir durch Wahl einer anderen Einheit in eine Integer-Zahl umwandeln. Beispielsweise können wir den Betrag 1,25 DM in 125 Pf umwandeln.
In einer der nächsten Folgen werden noch einige 32 Bit Arithmetik-Befehle vorgestellt. Diese erlauben durch geschickte Programmierung eine höhere Genauigkeit, als sie in Basic vorhanden ist.
Die *FORTH-Schreibweise* verschiedener Rechenaufgaben:
Aufgabe: 4 + ⟨17 * 12⟩
17 12 * 4 + . ⟨RETURN⟩ 208 OK
Durch das Stack-Konzept werden Klammern überflüssig.
Aufgabe: ⟨3 + 9⟩ * ⟨4 + 6⟩
3 9 + 4 6 + * . ⟨RETURN⟩ 120 OK
Aufgabe: 45 / 5 * 100
45 5 / 100 * . ⟨RETURN⟩ 900 OK
Weitere Arithmetik Befehle sind:
MOD [u^1 u^2 - -> Rest]
Dividiere u^1 durch u^2 und lege Rest auf Stack
11 3 MOD .⟨RETURN⟩ 2 OK
8 10 MOD .⟨RETURN⟩ 8 OK
/ MOD [u^1 u^2 - -> Rest Quotient]
Dividiere u^1 durch u^2 und lege Rest sowie Quotient auf Stack
11 3 / MOD .⟨RETURN⟩ 3 2 OK
8 10 / MOD .⟨RETURN⟩ 0 8 OK
* / [n^1 n^2 n^3 - -> n^4]
Errechne den Verhältnis-Faktor $n^4 = n^1 * n^2 / n^3$. Das Produkt $n^1 * n^2$, ist als Zwischenergebnis 32 Bit lang. Hierdurch ist eine höhere Rechengenauigkeit gewährleistet, als durch die Se-

quenz n^1 n^2 * n^3 / erreichbar wäre. Diese Anweisung wird häufig zum Rechnen mit genauen Konstanten verwendet. Folgende Konstanten können durch das Verhältnis n^2 / n^3 errechnet werden:

Konstante	Richtiger Wert	Verhältnis n^2 / n^3
PI	3.1415926	355 / 113
Wurzel aus 2	1.4142135	19601 / 13860
Wurzel aus 3	1.7320508	18817 / 10864

Einige Anwendungsbeispiele:

10355113 * / . ⟨RETURN⟩ 31 OK
Entspricht 10 * PI
100355113 * / . ⟨RETURN⟩ 314 OK
Entspricht 100 * PI

Wie aus dem Beispiel zu erkennen ist, wird mit diesem Befehl die größte Genauigkeit, die Integer-Zahlen bieten, erzielt.

* / MOD [n^1 n^2 n^3 – –> Rest n^4]
Dieser Befehl entspricht dem Vorangehenden, mit der Ausnahme, daß auch der Rest der Division auf den Stack gelegt wird.

10355113 * / MOD . .
⟨RETURN⟩ 3147 OK
100355113 * / MOD . .
⟨RETURN⟩ 31418 OK

ABS [n^1 – –> u^1]
Dieser Befehl wandelt n^1 mit Vorzeichen in die vorzeichenlose Zahl u^1 um. In Basic gibt es den gleichen Befehl.

– 123 ABS .⟨RETURN⟩ 123 OK
– 1 ABS . ⟨RETURN⟩ 1 OK
5 ABS . ⟨RETURN⟩ 5 OK

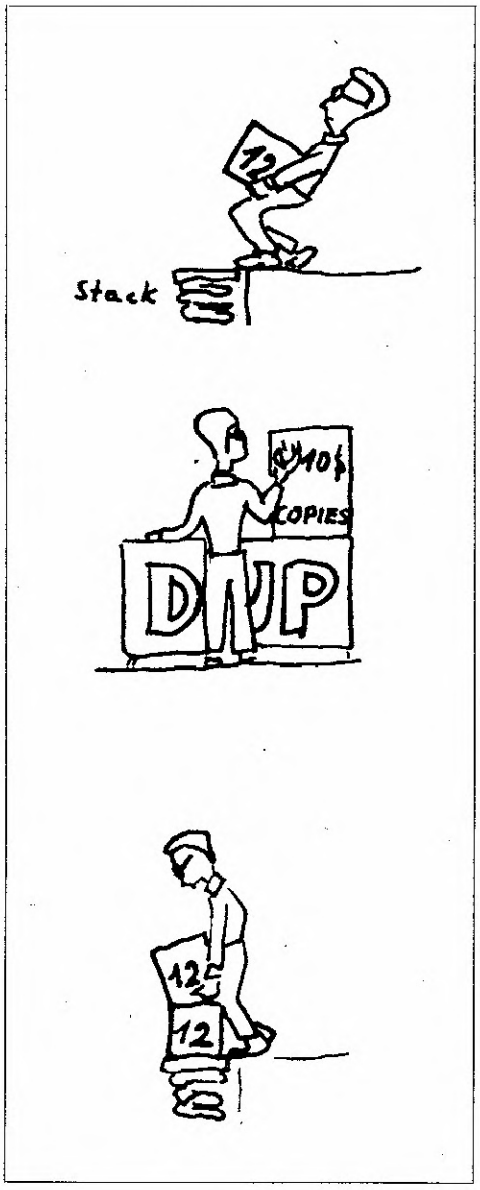
Häufig benutzte Befehls-Folgen, wurden in *FORTH* zu einem Befehl zusammengefügt:

1 + [n^1 – –> n^2] entspricht der Befehlsfolge 1 +
1 – [n^1 – –> n^2] entspricht der Befehlsfolge 1 –
2 + [n^1 – –> n^2] entspricht der Befehlsfolge 2 +
2 – [n^1 – –> n^2] entspricht der Befehlsfolge 2 –
2 * [n^1 – –> n^2] entspricht der Befehlsfolge 2 *
2 / [n^1 – –> n^2] entspricht der Befehlsfolge 2 /

Diese Befehls-Kombinationen sind nicht in allen *FORTH-Versionen* enthalten, können jedoch sehr leicht definiert werden. Das Definieren von Befehlen wird Thema des nächsten Teiles werden.

Stack Manipulation

Wir wir schon bei einigen Befehlen bemerkt haben, ist es wichtig, daß die Parameter eines Befehls immer in der richtigen Reihenfolge auf dem Stack liegen. Bei umfangreichen Befehlsfolgen muß deshalb oft die Reihenfolge



der Operanten geändert werden, um eine anschließende Operation zu ermöglichen. Um dies zu erreichen, werden folgende Befehle verwendet:

[n^1 – –> n^1 n^1]
DUP ist der am häufigsten verwendete Befehl. Man kann ihn sich sehr leicht merken, da DUP für Duplizierung steht. Er hat eine einfache, jedoch wichtige Aufgabe. Dieser Befehl dupliziert die oberste Zahl des Stacks.

100 DUP . . . ⟨RETURN⟩ 0010 OK
12 DUP . . ⟨RETURN⟩ 1212 OK
– DUP [n^1 – –> n^1 n^1]
Dieser Befehl entspricht dem DUP-Befehl, mit der Ausnahme, daß zuvor getestet wird, ob die zu duplizierende Zahl von Null verschieden ist. Nur wenn dies der Fall ist, erfolgt ihre Duplizierung.

100 – DUP . .⟨RETURN⟩ 010 OK
12 – DUP . .⟨RETURN⟩ 1212 OK
DROP [n^1 – –> empty]
ist das einfachste Befehlswort, welches lediglich die oberste Zahl des Stacks löscht.

100DROP . ⟨RETURN⟩ 10 OK
12 DROP . ⟨RETURN⟩ Fehlermeldung: Stack leer
SWAP [n^1 n^2 – –> n^2 n^1]
Dieser Befehl verändert ganz einfach die Anordnung der beiden obersten Zahlen. Dies ist oft vor Arithmetik-Operationen erforderlich.

100 SWAP . .⟨RETURN⟩ 100 OK
334 SWAP . .⟨RETURN⟩ 334 OK
OVER [n^1 n^2 – –> n^1 n^2 n^1]
Dieser Befehl arbeitet ähnlich wie der DUP-Befehl, jedoch wird hier der »Second« [d. h. die vorletzte Eintragung in den Stack] kopiert.

100 OVER . .⟨RETURN⟩ 10010 OK
334 OVER . .⟨RETURN⟩ 33433 OK
ROT [n^1 n^2 n^3 – –> n^2 n^3 n^1]
Hiermit wird die Anordnung der höchsten drei Stackeintragungen in zyklischer Rotation verändert. Dabei landet der älteste Eintrag an oberster Stelle.

102030 ROT . . .
⟨RETURN⟩ 103020 OK
102030 ROT ROT . . .
⟨RETURN⟩ 201030 OK
102030 ROT ROT ROT ROT . .
⟨RETURN⟩ 302010 OK

Einige erweiterte *FORTH-Versionen*, wie *64 FORTH* oder *FORTH 79*, besitzen noch einige, zusätzliche Stack-Manipulations-Befehle. Einen gewissen Standard besitzen folgende Befehle:

DEPTH [empty – –> n^1]
Es wird die Anzahl der auf dem Stack liegenden Parameter übergeben.

1020 DEPTH...
⟨RETURN⟩ 22010 OK
DEPTH .⟨[RETURN⟩ 0 OK
PICK [n^1 – –> n^2]
Die »n-te« Zahl wird auf den Stack gelegt. 1 PICK entspricht DUP; 2 PICK entspricht OVER

12434334 PICK . . . .
⟨RETURN⟩ 123334412 OK
1232 PICK . . .
⟨RETURN⟩ 12312 OK
ROLL [n – –>]
Dieser Befehl rotiert die »n-te« Zahl des Stacks an die oberste Stelle. 2 ROLL entspricht SWAP; 3 ROLL entspricht ROT.

121314154 ROLL . . . .
⟨RETURN⟩ 12151413
121314153 ROLL . . . .
⟨RETURN⟩ 13151412

Definition neuer Befehls-Worte

Nachdem wir uns mit den Arithmetik-Operationen beschäftigt haben, wollen wir uns nun mit der Definition neuer Befehle befassen. Die bislang vorgestellten *FORTH-Befehle* sind bis jetzt ausschließlich interaktiv und interpretativ aufgerufen worden. Das bedeutet, daß ein Wort (z. B. DUP oder +) mit der Eingabe seines Namens in die Tastatur aufgerufen wird, indem der äußere Interpreter dann das Wörterbuch (Dictionary) nach diesem Wort absucht, es lokalisiert und ausführt. Diese Art der Befehls-Abarbeitung wird im Gegensatz zu Basic nur bei der direkten Eingabe von Befehlen verwendet. Wir haben nun bereits so viele Befehle kennengelernt, daß wir daran gehen können, kleinere Programme damit zu erstellen. Wie schon im ersten Teil erwähnt, ist das gleichbedeutend damit, die zu anfangs erwähnten Befehle zu neuen Befehlen zusammenzusetzen.

Wollen wir einen Befehl definieren, so müssen wir als erstes mit dem Befehl »:«, den *FORTH-COMPILER* einschalten. Als nächstes wird die Bezeichnung und danach die eigentliche Definition des neuen Befehls eingegeben. Mit dem Befehl »:« wird der *FORTH-COMPILER* wieder ausgeschaltet und damit die Definition abgeschlossen.

Wir wollen nun einen Befehl definieren, welcher eine Zahl mit sich selbst multipliziert und das Ergebnis auf den Stack legt. Als Bezeichnung wählen wir »QUADRO«.

: QUATRO DUP ☆ ; ⟨RETURN⟩ OK

Wie zu erkennen ist, besteht die eigentliche Definition aus einer unbegrenzten Zahl von Befehlen. Der Befehl DUP dupliziert die oberste Zahl des Stack und der nachfolgende »☆« Befehl bildet, durch die Multiplikation, die Quadratzahl.

In den nachfolgenden Beispielen können wir den neuen Befehl testen:

2 QUATRO · ⟨RETURN⟩ 4 OK
5 QUATRO · ⟨RETURN⟩ 25 OK
10 QUATRO · ⟨RETURN⟩ 100 OK

Durch den Befehl »VLIST« können wir das gesamte Wörterbuch (Befehl's-Liste) auf Drucker oder Monitor ausgeben lassen. Der neue Befehl »QUATRO« steht nun am Anfang dieser Liste.

Bei der Definition von neuen Befehlen muß darauf geachtet werden, daß die Befehlsbezeichnung keine 31 Zeichen übersteigt. Es ist nicht unbedingt notwendig und auch nicht immer möglich, daß die Definition in eine einzige Zeile geschrieben wird. Als Beispiel wollen wir einen Befehl definieren, welcher die Kubikzahl eines Operanten errechnet:

: KUBIK DUP DUP ☆ ☆ ; ⟨RETURN⟩ OK

Wir können das gleiche auch wie folgt eingeben:

```
: KUBIK      ⟨RETURN⟩
  DUP DUP    ⟨RETURN⟩
  ☆ ☆        ⟨RETURN⟩
  ;          ⟨RETURN⟩ OK
```

Bei der direkten Eingabe von Befehlen ist die Schreibweise kaum von Bedeutung, da die Struktur der Eingabe nicht gespeichert wird. Ganz anders ist dies bei der Programmierung mit dem Texteditor. Hier sollte man unbedingt auf Strukturierung und gute Dokumentation achten, um auch später noch Erweiterungen oder Änderungen vornehmen zu können. Wir werden uns später noch genauer mit diesem Thema auseinandersetzen.

Nun wollen wir auch unseren KUBIK-Befehl testen:

2 KUBIK · ⟨RETURN⟩ 8 OK
5 KUBIK · ⟨RETURN⟩ 125 OK
10 KUBIK · ⟨RETURN⟩ 1000 OK

Werden in einem Programm oft die gleichen Befehls-Folgen verwendet, so ist es sinnvoll, diese zu einem Befehl zusammenzufassen:

: 1+ 1+ ; ⟨RETURN⟩ OK
: 1– 1– ; ⟨RETURN⟩
: 2+ 2+ ; ⟨RETURN⟩

Einige dieser einfachen Befehls-Folgen sind im *FORTH-Standard* schon enthalten und wurden im letzten Teil schon vorgestellt. Es ist nun sehr leicht, die fehlenden Kombinationen zu definieren. Wird in *FORTH* ein Befehl zweimal definiert, so gilt immer die letzte Defination.

FORTH-Befehle können auch zur Umsetzung dienen. Bei den meisten farbigen Computern dient ein Zahlencode zur Bestimmung einer Farbe.

Farbcode des Commodore 64:
0 = Schwarz
1 = Weiß
2 = Rot
3 = Türkis
usw.

Wollen wir statt des umständlichen Codes einfach die Farbe eingeben, können wir dies durch die folgenden Definitionen erreichen:

```
: SCHWARZ 0 ;  ⟨RETURN⟩
: WEISS   1 ;  ⟨RETURN⟩
: ROT     2 ;  ⟨RETURN⟩
: TÜRKIS  3    ⟨RETURN⟩
usw.
```

Die Funktion ist sehr einfach: Nach Eingabe der Farbbezeichnung legt der entsprechende Befehl den dazugehörigen Wert auf den Stack.

Es ist auch möglich, mit den Farben zu rechnen:

WEISS ROT + . ⟨RETURN⟩ 3 OK

Mit Hilfe eines PRINT-Befehls können wir auch leicht eine Vokabel-Übersetzung realisieren. Der PRINT-Befehl besitzt in *FORTH* folgende Syntax: ."text".

Elementare String-Befehle

Leider existiert in *FORTH* nur ein kleiner Befehlssatz zur Stringverarbeitung. Der *FORTH-Kern* enthält jedoch einige Worte für einfache Ein- und Ausgaben von Strings (Texten), die für viele Applikationen völlig ausreichen dürften.

»CR«
Dieser Befehl bewirkt auf dem Ausgabegerät (normal, Monitor) Zeilenvorschub und Wagenrücklauf.
12 13 14 . . CR . (return) 14 13
12 OK
. "TRONIC· CR . „VERLAG" (RETURN) TRONIC
Verlag OK

SPACE
Ein SPACE-Zeichen wird ausgegeben
. „APRIL" SPACE . „MAI" (RETURN)
APRIL MAI OK

Spaces (n1 –» empty)
Eine bestimmte Anzahl von SPACE-Zeichen wird ausgegeben.
. „APRIL" 4 SPACE . „MAI" (RETURN) APRIL MAI OK

EMIT (c –» empty)
Der ASCII-Charakter c wird ausgegeben. In Basic lautet dieser Befehl CHR$ (x) .
65 EMIT (RETURN) A OK
66 EMIT (RETURN) B OK

EXPECT (adr n –» empty)
Dieser Befehl entspricht in etwa dem INPUT des Basic-Interpreters. Ein String mit ‚n'-Zeichen wird von der Tastatur erwartet und ab Adresse ‚adr' abgespeichert. Ein RETURN kann diese Prozedur beenden, auch wenn die Anzahl ‚n' noch nicht erreicht sein sollte.
1ØØØØ 11 EXPECT (RETURN)
COMPUTRONIC (RETURN) OK

TYPE (adr n –» empty)
Dieser Befehl gibt eine Zeichenkette, die als Adresse ‚adr' gespeichert und ‚n'-Zeichen lang ist, auf dem Ausgabegerät aus.
1ØØØØ 11 TYPE (RETURN) COMPUTRONIC OK

COUNT (adr1 –» adr2 n)
Dieser Befehl erwartet eine Adresse ‚adr 1' auf dem Stack und ermittelt hieraus die Startadresse und die Zeichenzahl des Strings. Dies sind die Angaben, die von Type erwartet werden.

Die Anwendung dieses Wortes erfolgt also üblicherweise in der Form
‚adr1 COUNT TYPE',
wobei als ‚adr1' die Adresse des COUNT-Bytes (Zeichenanzahl) übergeben wird. Der eigentliche String muß dem COUNT-Byte folgen. KEY (–» c)
Dieser Befehl wartet auf eine beliebige Tastatureingabe und übergibt den entsprechenden ASCII-Code auf den Stack. Bei einigen Forth-Versionen ist die RETURN-Taste zur Quittierung erforderlich.

```
KEY . (RETURN)
A
65 OK
KEY KEY . . (RETURN)
A B
66 65 OK
KEY EMIT (RETURN)
D
D OK
```

Forth-Editor

Wir haben das Definieren von neuen Befehlswörtern kennengelernt. Wir haben auch gelernt, daß in *FORTH* definieren gleichbedeutend mit Programmieren ist. Um jedoch größere Programme entwickeln zu können, müssen wir diese abspeichern, laden und editieren können. Dies geschieht in *FORTH* – im Gegensatz zu Basic – mittels Texteditor. Der Texteditor ist leider in jeder *FORTH*-Version unterschiedlich zu handhaben. Wir wollen uns deshalb mit unserer Beschreibung auf den weitverbreiteten *„FIG-STANDARD"* beziehen.
Im Gegensatz zu vielen anderen Texteditoren wird in *FORTH* immer nur eine Bildschirmseite editiert. Man spricht dabei von einem „SCREEN". Diese SCREENs werden mit einer Nummer versehen und in der Regel auf Diskette gespeichert. Je nach *FORTH-Version* können 20 bis über 100 solcher Screens abgespeichert werden. Ein solcher Screen ist in 16 Zeilen zu je 64 Zeichen aufgeteilt. Da bei einigen Computern nur 40 Zeichen darstellbar sind, wird durch ein Links/rechts-Scrollen ein größerer Bildschirm simuliert. Ein Beispiel dafür ist *„64 FORTH"* von HES. Die meisten *FORTH-Versionen* besitzen mehrere TEXTPUFFER. Diese erlauben das gleichzeitige Einladen mehrerer „SCREENs".
Wollen wir einen Screen auf dem Bildschirm ausgeben lassen, so können wir dies mit dem Befehl: LIST.
Die Zeilennummern dienen nur zur Orientierung und werden deshalb nicht von allen *FORTH-Versionen* ausgegeben. Die erste Zeile sollte immer einen Kommentar enthalten, um nicht den Überblick zu verlieren. Der Kommentar beginnt mit einem (Zeichen, gefolgt von einem Leerzeichen und geht bis zum Zeichen). In diesem Kommentar sollte das Datum und die Aufgabe des SCREENs beschrieben werden. Selbstverständlich können auch in anderen Zeilen noch Kommentare notiert werden.
Wir haben nun gesehen, wie ein beschriebener SCREEN aussehen kann und wollen nun daran gehen, einen SCREEN zu beschreiben.
Zuerst suchen wir uns mit Hilfe eines Handbuches einen freien SCREEN aus. Mit ... LIST können wir uns überzeugen, daß der SCREEN leer ist. Ist beispielsweise der SCREEN Nummer 10 unbeschrieben, so geben wir folgenden Befehl ein: 10 EDIT (RETURN).

Nun befinden wir uns im eigentlichen Editor. Dieser ist im FIG-STANDARD zeilenorientiert. Das bedeutet, daß der Text mit Hilfe mehrerer Kurzbefehle editiert bzw. erstellt wird.

Die folgende Aufstellung zeigt die Funktion der Standard-Befehle:
LINE n –-» adr

Dieser Befehl übergibt dem Stack die Adresse der Zeilennummern. Ist der

entsprechende Screen nicht im Puffer, so wird dieser von der DISK eingelesen.
– MOVE adr --» n
Die Textzeile ab Adresse adr wird in Zeile n kopiert.
H n --»
HOLD übernimmt die Textzeile n nach PAD und hält sie dort zur Ausgabe bereit.
E n --»
ERASE löscht Zeile Nummer n und füllt diese mit Leerzeichen (Blanks) aus.
S n --»
Dieser Befehl fügt vor Zeile n eine Leerzeile ein. Nachfolgende Zeilen werden um eine Zeile nach unten verschoben. Die unterste Zeile geht verloren.
D n -»
Die Zeile n wird gelöscht. Dadurch wird der nachfolgende Text um eine Zeile angehoben.
T n --»
Zeile Nummer n wird auf dem Schirm ausgegeben.
L n --»
Der zur Zeit bearbeitete Screen wird ausgelistet.
R n --»
Dieser Befehl ist das Gegenstück zum H-Befehl. Die im PAD abgelegte Zeile wird in Zeile n ausgegeben.
P n --»
Der folgende Text wird in Zeile n übertragen. Der alte Zeileninhalt wird überschrieben.
I n --»
Der PAD wird ab Zeile n ausgegeben. Die nachfolgenden Zeilen werden um eine Zeile nach unten verschoben.
CLEAR n --»
Der Screen n wird mit Leerzeichen (Blanks) vollgeschrieben und somit gelöscht.
COPY n1 n2 --»
Dieser Befehl kopiert den gesamten Screen n1 nach Screen n2.
Die nachfolgend aufgestellten Befehle gehören zwar nicht zum Editor, sind jedoch zum Verwalten der Screens erforderlich.
LIST n1 --»
Screen n1 wird auf dem Schirm ausgelistet.
FLUSH
Alle geänderten Screens werden auf Diskette abgespeichert.
EDIT n1 --»
Screen n1 wird eingeladen und kann editiert werden.
LOAD n1 --»
Dies ist der wichtigste Befehl. Screen n1 wird in den Puffer eingeladen und dann Zeile für Zeile kompiliert. Wird ein Wort mehrmals definiert, so wird dies mit einer Meldung angezeigt. Der Kompilier-Vorgang wird in diesem Falle jedoch nicht unterbrochen. Um jedoch unnötigen Speicherverlust zu vermeiden, sollte man das Wörterbuch vor jedem neuen Kompilier-Vorgang zurücksetzen. Wie wir wissen, wird dies mit dem Befehl FORGET . . . gemacht.

Sicher wird sich so mancher schon gefragt haben, wie er längere Programme schreiben soll, wenn doch immer nur ein Screen kompiliert wird. Des Rätsels Lösung ist ganz einfach:
Mit Hilfe des --»-Befehls kann man beliebig viele Screens miteinander verbinden und somit sehr lange Programme schreiben.

Der --»-Befehl wird in diesem Fall an das Ende des letzten Screen's gesetzt. Aufgerufen wird jeweils nur der erste Screen. Je nach Datenträger und Anzahl der Puffer dauert der Kompiliervorgang bis zu einigen Minuten.

Einige *Forth-Versionen* besitzen auch die Möglichkeit, das Wörterbuch direkt abzuspeichern.

Im *C64 FORTH* von *HES* heißt der Befehl zum Abspeichern des Wörterbuches: *DSAVE*.

Das Einladen geschieht in diesem Fall mit DLOAD. Das D steht in beiden Fällen für Dictionary und ist gleichbedeutend mit Wörterbuch. Bei der Benutzung dieser beiden Befehle sollte man bedenken, daß nur der compilierte Code angespeichert wird und somit keine Änderungen mehr möglich sind.
Wir haben nun eine Menge über den *FORTH-Texteditor* erfahren und können somit auch längere Programme schreiben, ohne diese nach dem Ausschalten zu verlieren.

Wie schon erwähnt, ist es von größter Wichtigkeit, seine *FORTH-Programme* gut zu dokumentieren, da *FORTH-Befehle* allein sehr unübersichtlich wirken.

Zur Dokumentation ist übrigens an dieser Stelle zu sagen, daß Kommentare unbestreitbar den Vorzug haben, unverrückbar physikalisch an die kommentierte Definition geknüpft zu sein. Daher wird jede Änderung eines Programms auch mit der Aktualisierung der Kommentierung zwangsläufig verbunden sein. Dennoch sollte man die Kommentare auf der Diskette knapp halten. Genauer:
Die mit dem Code verknüpften Kommentare sollten knapp sein, da sonst die Kompilierungszeit unnötig erhöht wird.
Daneben ist die Lesbarkeit von Texten innerhalb von Codes relativ schlecht und verschlechtert auch die Lesbarkeit des Codes.
Da die Diskettenlaufwerke von Heimcomputern auch keine unbegrenzte Speicherkapazität besitzen, sollte man sich überlegen, ob man die Dokumentation nicht wie bei Assembler-Programmen getrennt vom Programm vornimmt.
Wie wäre es denn z. B. mit Papier und Bleistift?

Bedingte Verzweigungen
Das Forth-Befehlswort „IF" eröffnet alle bedingten Verzweigungen in FORTH-Programmen. Dieses Befehlswort erwartet eine 16-Bit Zahl auf dem Stack und behandelt dieses als Flag. Der „IF"-Befehl arbeitet nur zusammen mit dem Befehlswort „ENDIF". „ELSE" ist ebenfalls möglich, jedoch nicht unbedingt nötig. Folgendes Beispiel zeigt den Zusammenhang der einzelnen Befehle:

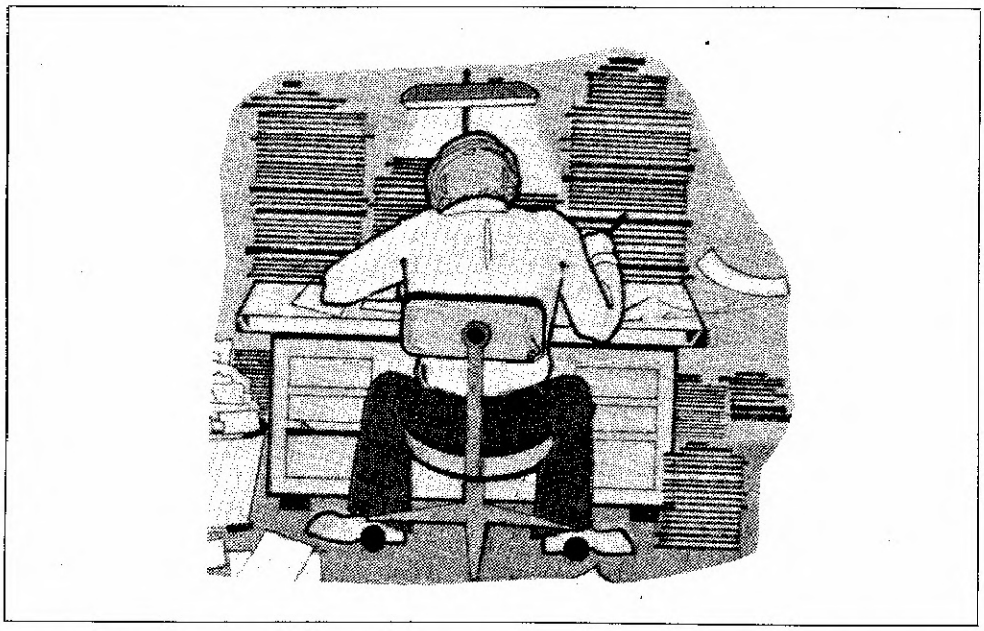
```
IF      (IF testet das Flag; falles es
        wahr (= 1) ist ...)

...     (werden die nebenstehenden
        Anweisungen ausgeführt

...     (ansonsten)

ENDIF(fortgefahren)

IF
...     (falls Flag = 1, werden neben-
        stehende Worte)

...     (bis ELSE ausgeführt)

ELSE
...     (andernfalls erfolgt die Ausfüh-
        rung der Worte)
...     (zwischen ELSE und ENDIF)

ENDIF
```

Bei der Verwendung von Verzweigungen sollte man immer bedenken, daß diese nicht interprevativ verwendet werden dürfen, also nur innerhalb einer Definition. Probieren wir doch einmal die folgende, einfache Definition:

```
:DIVISION(Division mit Kommentar,
          falls Rest nicht Null) (Para-
          meter wie bei / – Befehl)

/MOD.     (Division mit Rest)

SWAP      (Bringe Rest zum TOS und
          teste ihn als Flag)

„IFDIVISION geht nicht auf"

ENDIF
;         (falls Rest = „wahr", wird
          der Text gedruckt)
```

Das nun definierte Wort „DIVISION" erwartet also zwei Zahlen auf dem Stack, dividiert diese und beläßt den Quotienten auf dem TOS. Die Syntax des neuen Befehls entspricht genau dem Original "/" Befehl. Im Gegensatz zu diesem, wird jedoch bei Entstehung eines Restes die Meldung „DIVISION GEHT NICHT AUF", ausgegeben.
Wir wollen nun ein Beispiel zeigen, welches den „ELSE"-Befehl vorführt. Es handelt sich hier um einen Vergleichs-Befehl, welcher angibt, ob die erste Zahl kleiner oder größer als die zweite Zahl ist.

Da im Standard-Basic kein „ELSE"-Befehl zugelassen ist, muß man sich mit zwei „IF" Anweisungen behelfen, wenn man nicht zu dem UNSAUBEREN „GOTO"-Befehl greifen will. Dies geht selbstverständlich auf Kosten der Zeit.
Im übrigen sei an dieser Stelle darauf hingewiesen, daß einige *FORTH-Versionen* statt des Befehles „ENDIF" den Befehl „THEN" verwenden. Dies muß in dem entsprechenden Manual nachgelesen werden.
Wir haben nun einiges über die „IF"-Anweisung gelernt und wollen uns



```
:VERGLEICHE (Vergleich zweier
            Zahlen)
<           ."(FLAG ermitteln)
IF          ."KLEINER"
ELSE        ."GROESSER"
ENDIF
;
```

Beispiel:
33 100 VERGLEICHE (RETURN)
KLEINER OK
100 33 VERGLEICHE (RETURN)
GROESSER OK

Zum Vergleich dasselbe noch einmal in der Programmiersprache Basic:

```
10 INPUT „ZAHL1";Z1
20 INPUT „ZAHL2";Z2
30 IF Z1<Z2 THEN PRINT „KLEINER"
40 IF Z1>Z2 THEN PRINT „GROESSER"
50 END
```

nun die eigentlichen Vergleichsbefehle näher ansehen:

< (n1 n2 n3 --> n1 f)
Flag wird 1 wenn die Zahl n2 kleiner als n3 ist.

> (n1 n2 n3 --> n1 f)
Flag wird 1 wenn die Zahl n2 größer als n3 ist.

= (n1 n2 n3 --> n1 f)
Flag wird 1 wenn beide Zahlen gleich sind

0< (n1 n2 n3 --> n1 n2 f)
Flag wird 1 wenn n3 eine negative Zahl ist

0= (n1 n2 n3 --> n1 n2 n3)
Flag wird 1 wenn die Zahl n3 gleich null ist.

Als Ergebnis des Vergleiches wird immer eine Boolsche Zahl abgelegt. Diese wird auch als Flag bezeichnet und gibt an, ob die Aussage richtig oder falsch ist. Ist diese richtig, so wird eine 1, ansonsten eine 0 auf den Stack gelegt.

Beispiele:

23 30 < . <RETURN> 1 OK
30 23 < . <RETURN> 0 OK
30 44 = . <RETURN> 0 OK
-2 0 < . <RETURN> 1 OK

Bei dem Vergleich zweier Zahlen sollte man bedenken, daß diese vom Vergleichsbefehl überschrieben werden. Will man diese jedoch für weitere Berechnungen erhalten, so kann man dies mit dem Befehl OVER erreichen:
23 30 OVER OVER < . . . <RETURN> 1 30 23

Programmschleife DO...LOOP

Ein wichtiger Bestandteil aller Programmiersprachen sind Schleifen. Unter Schleife versteht man einen oder mehrere Befehle, die durch zwei Kennzeichnungen eingeklammert sind. Diese eingeklammerten Befehle werden beliebig oft durchlaufen. Ist eine bestimmte Anzahl durchlaufen, wird das Programm fortgesetzt. In *FORTH* gibt es genau wie in Pascal keinen „GOTO" Befehl, deshalb ist der Programmierer auf die Schleife angewiesen.

Der Schleifenbefehl hat in *FORTH* folgende Syntax:

DO ... LOOP (n1 n2 n3 --> n1)
Die Schleife wird von n3 bis n2 in Einerschritten durchlaufen. Die Zahl n3 muß in diesem Fall kleiner als n2 sein.

Die oberste Zahl auf dem Stack wird immer als Anfangswert und die darunterliegende Zahl als Endwert verwendet. Während der Schleife wird der Zahlenparameter immer um eins erhöht, deshalb muß der Endwert immer größer als der Anfangswert sein. Ist der Zahlenparameter größer oder gleich dem Endwert, so wird die Schleife verlassen. Da diese Kontrolle jedoch immer erst am Ende einer Schleife durchgeführt wird, ist die Schleife bereits einmal durchlaufen.
Will man während eines Schleifendurchlaufs den Zahlenparameter auslesen, kann man dies mit Hilfe des Wortes „I". Dieses Wort legt den Zahlenparameter auf den Stack.

Wie der „IF" Befehl kann auch die DO-LOOP-Schleife nicht interpretativ ausgeführt werden und muß deshalb innerhalb einer Definition stehen.
Das folgende Beispiel gibt die Zahlen 0–12 auf dem Schirm aus:

: ZAHLEN 12 0 DO I . LOOP ; <RETURN> OK

ZAHLEN <RETURN> 0 1 2 3 4 5 6 7 8 9 10 11 12 OK

Das gleiche in BASIC:
FOR I = 0 TO 12 : PRINT I, :NEXT 1

Ist der Endwert kleiner als der Anfangswert, so wird die Schleife nur einmal durchlaufen:

: ZAHLEN 0 12 DO I . LOOP ; <RETURN> OK
ZAHLEN <RETURN> 12 OK

Das nächste Beispiel zeigt zwei DO-LOOP-Schleifen die ineinander verschachtelt sind. Der Aufruf der neuen Definition gibt die Zahlen 0 bis 9 in einer Dreieckmatrix aus.

: SCHLEIFE1 0 DO I . LOOP ; <RETURN> OK

: SCHLEIFE2 0 DO CR 1 - DUP SCHLEIFE1 LOOP ; <RETURN> OK
11 10 SCHLEIFE2 <RETURN> OK

0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6
0 1 2 3 4 5
0 1 2 3 4
0 1 2 3
0 1 2
0 1
0 OK

In vielen Programmen werden DO-LOOP-Schleifen zur Verzögerung von Programmabschnitten verwendet. Durch den folgenden Vergleich wird ersichtlich, wie schnell die Programmiersprache *FORTH* arbeitet:

Eine Leerschleife in FORTH:
: PAUSE 10000 0 DO LOOP ;

Die Ausführungszeit dieser Schleife beträgt im Durchschnitt 4 Sekunden. Je nach *FORTH-Version* und Microprozessor kann diese Zeit etwas länger oder auch kürzer werden.

Die gleiche Schleife in BASIC:
10 FOR I = 0 TO 10000 : NEXT 1

In Basic ist die durchschnittliche Ausführungszeit etwa 4 bis 5 mal so lang wie in *FORTH*. Wie zu erkennen ist, eignet sich somit *FORTH* auch für Programme, in denen es auf genaue „Pause-Zeiten" ankommt. (Z. B. Eprom-Programmierung oder sonstige Ein- bzw. Ausgaben).

So mancher wird sich inzwischen fragen, ob man das Zahlenparameter nicht auch um 2, oder einer anderen Zahl erhöhen kann. Dies geschieht in Basic durch den Befehl „STEP".
Dies ist auch in *FORTH* möglich, indem man statt des „LOOP"-Befehls ein „+LOOP" einsetzt. In diesem Fall wird die Zahl, welche vor dem „+LOOP" auf dem Stack liegt, als Increment benutzt. Ist diese Zahl im negativen Bereich, so wird die Schleife bei jedem Durchlauf erniedrigt. In diesem Fall muß der Endwert kleiner als der Anfangswert sein, da sonst nur ein Durchlauf erfolgt.

Die Syntax: +LOOP (n1 n2 n3 --> n1 n2)

Einige Beispiele:

: ZAHLEN 10 0 DO I . 2 +LOOP ; <RETURN> OK

ZAHLEN <RETURN> 0 2 4 6 8 OK
In BASIC:

10 FORI = 0 TO 10 STEP 2 : PRINT I, : NEXT I

Umgekehrte Schleife:

: ZAHLEN 0 10 DO I . -2 +LOOP ; <RETURN> OK

ZAHLEN <RETURN> 10 8 6 4 2 OK

In Basic:

10 FORI = 10 TO 2 STEP -2 : PRINT I, : NEXT I

Bei Verwendung des „+LOOP" sollte man daran denken, daß vor dessen Ausführung eine 0 zur Endlosschleife führt.

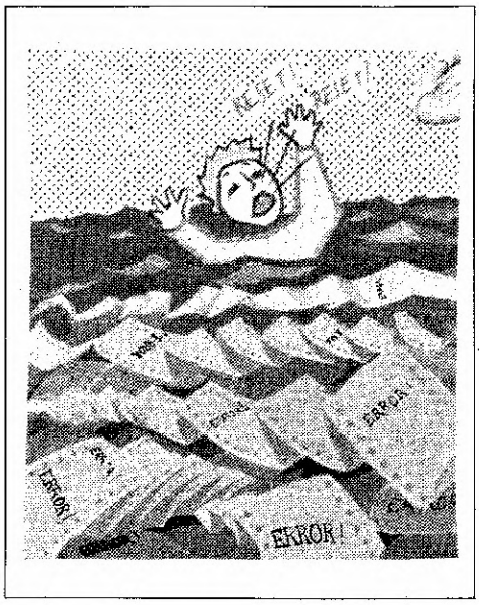
BEGIN...UNTIL-Schleife

Da in *FORTH* kein Sprungbefehl vorhanden ist, müssen einige spezielle Schleifen diese ersetzen. Eine davon ist die BEGIN-UNTIL-Schleife.
Diese Schleife wird so lange durchlaufen, bis vor dem Befehl „UNTIL" ein „WAHR-Flag" gesetzt wird. Eine 0 führt zu einem neuen Durchlauf. Diese BEGIN-UNTIL-Struktur kann also sehr leicht den „GOTO"-Befehl ersetzen und führt somit zu einem übersichtlicheren Programm.
Die Struktur der Schleife sieht wie folgt aus:

BEGIN
... (Befehle innerhalb der Schleife)
...
...
... (Testteil der Schleife z. B. Vergleich)
(Flag wird auf Stack gelegt)
UNTIL (Ist Flag=0 erfolgt Sprung nach Begin)
... (Flag ist ungleich 0; Programm wird fortgesetzt)
...

Das Beispiel mit den Zahlen:

: ZAHLEN
0
BEGIN
1+ (Zahl um 1 erhöhen)



DUP . (Zahl ausgeben)
DUP 10 = (Ist 10 erreicht?)
UNTIL
;
ZAHLEN <RETURN> 1 2 3 4 5 6 7 8 9 10 OK

In dem obigen Beispiel wird die Schleife so lange wiederholt, bis die auf dem Stack liegende Zahl den Wert 10 erreicht hat.

Eine weitere Anwendung wäre beispielsweise das Abfragen der Tastatur:

: INKEY (Auf <RETURN> wird gewartet)
BEGIN
KEY
13 = (Ist <RETURN> gedrückt)
UNTIL
"RETURN WURDE GEDRUECKT"
;
INKEY <RETURN>

<RETURN> RETURN WURDE GEDRUECKT OK

Zum Schluß das Beispiel noch einmal in Basic:

10 GET E$: IF E$="" THEN 10

20 IF ASC(E$(<> 13 THEN 10

30 PRINT „RETURN WURDE GEDRUECKT"

Die Endlosschleife Begin ... Again

Diese Programmschleife kennt keine Aussprungbedingung, ist also endlos. Die Hauptanwendung dieses Befehlswortes liegt im Aufruf bzw. in der unbegrenzt häufigen Abarbeitung eines (periodisch zu durchlaufenden) Hauptprogrammes. Die Syntax dieser Schleife entspricht weitgehend der BEGIN-UNTIL-Schleife.

Beispiel:

```
: TEST
    BEGIN
      ."COMPUTRONIC" CR (diese Anweisung wird endlos ausgeführt)
    AGAIN
;
```

Zu bemerken ist noch, daß es sich bei dieser Schleifenart um die schnellste *FORTH-Schleife* handelt.

BEGIN-WHILE-Schleife

Auch diese Schleifenstruktur wird beliebig lange durchlaufen. Im Gegensatz zur BEGIN-UNTIL-Schleife erfolgt die Abfrage einer Aussprungbedingung (Flag) am Anfang des Schleifenkernes. Ist die Bedingung am Schleifenbegin nicht erfüllt, so wird die Schleife nicht ein einziges Mal durchlaufen. Das folgende Beispiel zeigt die Syntax dieser Schleife:

```
BEGIN
...        (Bedingung zur Erzeugung des Flags)
WHILE      (solange das Flag = 1 ist,
...        wird der folgende Teil durchlaufen)
REPEAT     (Rücksprung nach BEGIN)
```

In dem folgenden Beispiel wird die Tastatur abgefragt. Ist die Taste „C" gedrückt, so wird der Text „TASTE C IST GEDRÜCKT" ausgegeben. Die Schleife wird erst verlassen wenn eine andere Taste gedrückt wird.

```
: TASTE
   BEGIN
     KEY 67 - 0 = (Ist Taste „C" gedrückt?)
       WHILE
         ."TASTE C IST GEDRÜCKT"
CR
       REPEAT (Springe zurück)
;
```

Wir haben nun alle verfügbaren *FORTH-Schleifen* kennengelernt. Es gibt in *FORTH* auch die Möglichkeit noch weitere Arten von Schleifen zu definieren, doch dies ist dem etwas erfahreren Programmierern vorbehalten. Wer sich genauer damit beschäftigen möchte, muß auf andere Literatur zurückgreifen, da die Beschreibung dieser Definitionen den Rahmen dieses Kurses sprengen würde.

FORTH-Programmbeispiele:

Berechnung des größten gemeinsamen Teilers zweier Zahlen:

```
: TEILER (Größter gemeinsamer Teiler errechnen)
    BEGIN
      SWAP OVER MOD DUP
      0 =
    UNTIL
    DROP
;
27 21 TEILER ‹RETURN› 3 OK
 8  6 TEILER ‹RETURN› 2 OK
```

Die Berechnung wurde mit Hilfe des Euklidschen Algorithmus verwendet. Zuerst bestimmt man den Rest ›R‹ zweier Zahlen, ›A‹ und ›B‹, durch (A MOD B). Ist R=0, dann ist B der gesuchte Teiler. Ist R‹›0, so wird A=B und B=R. Danach wir die obige Division wieder ausgeführt. Solange, bis die Bedingung R=0 erfüllt ist.

Variablen und Konstanten in Forth

Obwohl die meisten Zahlenwerte und Rechenoperationen über den Stack geführt werden, kann man trotzdem auch Konstante und Varibalen definieren.
In Basic wird eine Konstante wie folgt definiert:

```
10 LET Y = 10
```

Der Konstanten Y wird der Zahlenwert 10 zugeordnet. In *FORTH* würde das gleiche so geschrieben:

```
10 CONSTANT Y ‹RETURN› OK
```

Durch diese Befehlsfolge wird im Wörterbuch eine Konstante „Y" eingetragen, die den Zahlenwert 10 enthält. Sobald der Variablennamen aufgerufen wird, erhält der Stack den entsprechenden Zahlenwert.

Beispiele:

```
12 Y + . ‹RETURN›  22 OK
 5 Y ★ . ‹RETURN›  50 OK
 Y Y ★ . ‹RETRUN› 100 OK
```

Das ganze kann auch in einer Definition stehen:

```
: Test
    Y ★ .
;
7 TEST ‹RETURN› 70 OK
```

Wie wir wissen, können wir statt der Konstanten auch einfach folgendes schreiben:

```
: Y 10 ;
```

Dies hat zwar die gleiche Wirkung wie die Definition einer Konstanten, ist jedoch in der Ausführungszeit wesentlich länger.
Im Gegensatz zu den meisten BASIC-Versionen können die Namen der Variablen, Konstanten und Befehlen in den meisten *FORTH-Versionen* bis zu 31 Zeichen lang sein. Es ist so möglich, den Variablen einen sinnvollen Namen zu geben, um somit das Programm übersichtlicher zu machen.

Ähnlich wie die Konstanten werden in *FORTH* auch die Variablen definiert:

```
10 VARIABLE TAG
```

In diesem Beispiel wurde der Variablen „TAG" der Wert 10 zugewiesen. Der Unterschied zwischen Variablen und Konstanten besteht darin, daß die Konstante nach der Definition ihren Wert immer behält. Dieser wird beim Aufruf der Konstanten auf den Stack gelegt. „Beim Aufruf einer Variablen wird dagegen nicht der Wert der Variablen, sondern die Adresse dieser Variable auf den Stack gelegt. Zum Ändern bzw. Auslesen des Wertes einer Variablen, benötigen wir noch zwei weitere Befehle, nämlich den Befehl :Wert holen (Load) ; und den Befehl :Wert wegspeichern (STORE).

Für den STORE-Befehl wird in *FORTH* das „!" Zeichen verwendet. Dieser Befehl entspricht weitgehend dem bekannten POKE aus dem BASIC. Jedoch wird hier keine 8-Bit-Zahl, sondern eine 16-Bit-Zahl abgespeichert.
Die Syntax des STORE-Befehl:

```
! (n1 n2 adr --› n1)
```

Für den LOAD-Befehl wird in Forth das " " Zeichen verwendet. Dieser Befehl ist mit dem Peek-Befehl im Basic zu vergleichen. Auch hier wird eine 16-Bit-Zahl ausgelesen:

Die Syntax des LOAD-Befehl:

```
@(n1 n2 adr --› n1)
```

Die eben vorgestellten Befehle LOAD und STORE arbeiten nur mit 16-Bit-Zahlen. Will man dagegen nur ein einziges Byte, also eine Dezimalzahl zwi-

schen 0 und 255 abspeichern, so verwendet man das Wort C! „Mit Ce wird ein Byte auf den Stapel geholt. Dabei werden die oberen 8 Bits zu Null gesetzt.

Wenn wir nur den Variablennamen, gefolgt von einem Dezimalpunkt (Print) eingeben, so erhalten wir die Adresse der Variablen. Diese kann je nach *FORTH-Version* und Anzahl der Definitionen verschieden sein.

TAG . ‹RETURN› 4104 OK
Möchte man den Wert der Variablen TAG auslesen, so geschied dies wie folgt:

TAG @ . ‹RETURN› 10 OK

Das Ändern einer Variablen geht ebenso einfach:

13 TAG ! ‹RETURN› OK
(Der Wert 13 wird gespeichert)

TAG @ . ‹RETURN› 13 OK
(Der Wert wird ausgelesen)

Auf ähnliche Weise kann man auch den Wert einer schon vereinbarten Konstanten ändern. Die Adresse einer Konstanten erhält man mit dem Befehlswort "'" (Einzelapostroph).
Die Syntax:
' (Name) (n1 n2 --› n1 n2 adr)
Die Adresse der Konstanten (Name) wird auf den Stack gebracht.

Beispiele:

20 CONSTANT TRONIC ‹RETURN› OK
(der Konstanten TRONIC wird der Wert 20 zugewiesen)

TRONIC . ‹RETURN› 20 OK
(der Wert wird ausgelesen)

30 ' TRONIC ! ‹RETURN› OK
(der Wert 30 wird in die Konstante geschrieben)

TRONIC . ‹RETURN› 30 OK
(nochmal überprüfen)

FORTH-Zahlensystem

Bei der Initialisierung von *Forth,* wird immer das Rechnen zur Basis 10, also die gewohnte Dezimalrechnung angenommen. *Forth* kann jedoch auch mit allen anderen Zahlensystemen rechnen. Will man z. B. mit Hexadezimalzahlen arbeiten, so kann mit dem Befehl „HEX" auf diesen Modus umgeschaltet werden. Danach erfolgen alle Ein- und Ausgaben in hexadezimaler Schreibweise.

HEX (n1 n2 n3 --› n1 n2 n3)

Möchte man wieder in den Dezimal-Modus zurück, so geschieht dies mit dem Befehl "DECIMAL".

DECIMAL (n1 n2 n3 --› n1 n2 n3)

Beispiel:

DECIMAL 90 HEX . ‹RETURN› 5A OK
Die Dezimalzahl 90 entspricht der Hexadezimalzahl 5A.

HEX 5A DECIMAL . ‹RETURN› 100 OK
Das umgekehrte Beispiel.

Möchte man ein anderes Zahlensystem verwenden, muß man in die System-Variable BASE den entsprechenden Wert speichern. Durch den Befehl HEX wird eine 16 und durch DECIMAL eine 10 in die Variable geschrieben.

Möchten wir beispielsweise das Binärsystem programmieren, so müssen wir nur die Zahl 2 (Zustand 0 und 1) in die Variable BASE bringen. Den Umschalt-Befehl nennen wir BIN.

: BIN 2 BASE ! ;

Mit dem Befehl BIN können wir nun jederzeit auf Binär-Modus umschalten.

```
DECIMAL 255 BIN .
      ‹RETURN› 11111111 OK
DECIMAL 16 BIN .
      ‹RETURN›     10000 OK
BIN 01010101 DECIMAL .
      ‹RETURN›        85 OK
BIN 00010000 DECIMAL .
      ‹RETURN›        16 OK
```

Hier ein kleines FORTH-Programm

Berechnung der Primzahlen zwischen zwei Grenzen.

```
4 VARIABLE ZEILE
: TEST     MOD 0=;
: DRUCKE   DUP 4 .R ZEILE e DUP 0=
           IF CR DROP 4
           ELSE 1 -
           THEN
           DROP ;

: PTEST    DUP 2 / 2 DO DUP I TEST
           IF 0 LEAVE
           THEN LOOP DUP IF
           DRUCKE
           ELSE DROP
           THEN
           DROP ;

: PRIM     (ende anf --› )
CR 4       ZEILE !
DO I       PTEST LOOP CR ;

200 1 PRIM ‹RETURN›
```

1	2	5	7	11
13	17	19	23	29
31	37	41	43	47
53	59	61	67	71
73	79	83	89	97
101	103	107	109	113
127	131	137	139	149
151	157	163	167	173
179	181	191	193	197
199	OK			

Die Spezial-Schleifen

Die Programmiersprache *FORTH* enthält natürlich eine beachtliche Anzahl von Wörtern, die von den im letzten Teil vorgestellten Strukturelementen selber Gebrauch machen. Etliche von ihnen haben auch unter reinen Anwender-Gesichtspunkten große praktische Bedeutung.
Daneben existieren einige Worte, die im wesentlichen spezialisierte Maschinensprache-Schleifen aufrufen. Die typischen Vertreter dieser Gruppe sind:

CMOVE (adr1 adr2 n --› empty)
Die durch ›n‹ bestimmte Anzahl Bytes, werden von Speicherbereich adr1 nach Speicherbereich adr2 kopiert. Dabei wird so vorgegangen, daß zuerst die Bytes mit den niedrigsten Adressen, fortschreitend zu höheren Adressen, transportiert werden.

FILL (adr n1 n2 --› empty)
Der Speicherbereich ab Adresse adr wird mit n1 Bytes des Bitmusters n2 gefüllt.

ERASE (adr n --› empty)
Der Speicherbereich ab adr n wird mit n Bytes des Bitmusters 0 gefüllt. Also gelöscht!

BLANKS (adr n --› empty)
Die Speicherbereich ab adr n wird mit n Bytes des Bitmusters 32 (SPACE) gefüllt.

Diese Worte gehören naturgemäß zu den „schnellsten" *FORTH-Worten* überhaupt, da diese in Maschinensprache definiert sind.

weiter auf Seite 64

Fortsetzung von Seite 39

FORTH-Zahlensystem

In den letzten Teilen unseres Kurses haben wir uns ausschließlich mit dem ganzzahligen Zahlensystem beschäftigt. Wie bekannt, umfaßt dieses System den Zahlenbereich von –32768 bis +32767. Obwohl dieser Bereich nicht sehr umfangreich ist, reicht er doch für viele Anwendungen (z. B. Steuerungsaufgaben) völlig aus. Daß wir in *FORTH* auch mit längeren Zahlen arbeiten können, zeigt der folgende Abschnitt:

Doppelt lange Zahlen

Um in *FORTH* größere Zahlen zu ermöglichen, werden zwei normale Integer-Zahlen zu einer „Doppelten Zahl" zusammengefaßt. Durch diesen Trick steht jeder Zahl eine Matrix von vier Bytes zur Verfügung. Doppelt lange Zahlen umfassen somit den Zahlenbereich von –1.073741824 bis +1073741823.

Eine doppelt lange Zahl wird durch einen Dezimalpunkt gekennzeichnet. Dieser kann an jeder beliebigen Stelle sitzen und ist nicht mit dem üblichen Komma zu verwechseln, da dieser nur zur Kennzeichnung dient.

Beispiel:

Die gleiche Zahl verschieden geschrieben
1.234567
12.34567
123.4567
1234.567
12345.67
123456.7
1234567.

Da *FORTH* eigentlich für Steuerungsaufgaben entwickelt wurde und somit nur selten mit doppelt langen Zahlen gearbeitet wird, sind nur wenige Befehle vereinbart.

D. (d1 d2 d3 --› d1 d2)
Eine doppelt lange Zahl wird ausgegeben.
12345.67 D. ‹RETURN› 1234567 OK
1234.567 D. ‹RETURN› 1234567 OK

D+ (d1 d2 d3 --› d1 d)
Die Summe von d2 und d3 wird auf den Stack gelegt.

12000. 40000. D+ ‹RETURN› 52000 OK
90000. 5.000 D+ ‹RETURN› 95000 OK

D– (d1 d2 d3 --› d1 d)

Die Differenz von d2 und d3 wird auf den Stack gelegt.
50000. 10000. D– ‹RETURN› 40000 OK
55000. 44.000 D– ‹RETURN› 11000 OK

D.R (d1 d2 d3 n --› d1 d2)

Dieser Befehl entspricht weitgehend dem D.-Befehl. Druckt jedoch die Zahl rechtsbündig in ein Feld der Weite n. Das Feld (Fenster) n beginnt an der aktuellen Position des Ausgabegerätes. Einen ähnlichen Befehl kennen einige Basic-Interpreter unter der Bezeichnung ‚PRINT-USING'. Das folgende Beispiel zeigt die Leistungsfähigkeit dieses Befehles:

PRINT (6 Zahlen rechtsbündig ausgeben)
CR 7 D.R
CR 7 D.R
CR 7 D.R
CR 7 D.R
CR 7 D.R
CR 7 D.R
2. 20. 200. 2000. 20000. 200000.
‹RETURN› OK

PRINT ‹RETURN›
200000
20000
2000
200
20
2 OK

DABS (d --› ud)
Der Absolutwert von d wird auf den Stack abgelegt.
–1234567. DABS ‹RETURN› 1234567 OK
–102. DABS ‹RETURN› 102 OK

DMAX (d1 d2 d3 --› d1 d)

Die Zahlen d2 und d3 werden verglichen und durch die größere Zahl ersetzt.
12345. 13400. DMAX D. ‹RETURN› 13400 OK
15000. 14000. DMAX D. ‹RETURN› 15000 OK

DMIN (d1 d2 d3 --› d1 d)
Die Zahlen d2 und d3 werden verglichen und durch die kleinere Zahl ersetzt.

14010. 12000. DMIN D.‹RETURN› 12000 OK
14050. 55000. DMIN D.‹RETURN› 14050 OK

Vergleichsbefehle für doppelte Zahlen

D= (d1 d2 d3 --› d1 f)
Flag wird 1, wenn d2 gleich d3 ist

D0= (d1 d2 d3 --› d1 d2 f)
Flag wird 1, wenn d3 gleich 0 ist

D‹ (d1 d2 d3 --› d1 f)
Flag wird 1, wenn d3 kleiner als d2 ist

Gemischte Zahlenoperationen

Bei der Multiplikation von Integer-Zahlen entstehen oft Überträge, die eine Integer-Zahl nicht mehr fassen kann. Es wäre deshalb wünschenswert, wenn es einen Befehl gäbe, welcher aus einer 16 Bit-Multiplikation ein 32 Bit-Ergebnis, also eine doppelt lange Zahl, bildet. Genau dieses gestattet der Befehl M★.

M★ (n1 n2 --› d1)
Aus dem Produkt zweier Integer-Zahlen wird eine doppelte Zahl gebildet.

Beispiele:
100 500 M★ D. ‹RETURN› 50000 OK
90 500 M★ D. ‹RETURN› 45000 OK
400 400 M★ D. ‹RETURN› 160000 OK

Das ganze funktioniert auch umgekehrt. Dividieren wir eine 32 Bit-Zahl durch eine 16 Bit-Zahl so erhalten wir eine 16 Bit-Integer-Zahl. Auch für diesen Fall gibt es einen Befehl: M/

M/ (d1 n1 --› n1)
Eine doppelte Zahl dividiert durch eine Integer-Zahl ergibt eine Integer-Zahl.

Beispiele:
50000. 20 M/. ‹RETURN› 2500 OK
100000. 50 M/. ‹RETURN› 2000 OK
50000. 25000 M/ . ‹RETURN› 2 OK

Zahlen ohne Vorzeichen

Normalerweise verwendet *FORTH* zum Abspeichern von Integer-Zahlen immer nur 15 Bit. Das 16. Bit kennzeichnet, welches Vorzeichen vorhanden ist, Plus oder Minus. Eine Integer-Zahl kann somit den Bereich von –32768 bis 32767 umfassen. Verzichtet man auf das Vorzeichen, so würde der Zahlenbereich von 0 bis 65535 gehen. Diese Möglichkeit bietet der Befehl U.

U. (n1 n2 n3 --› n1 n2)
Die Zahl n3 wird ohne vorzeichen ausgegeben.

1000 U. ‹RETURN› 1000 OK
30000 U. ‹RETURN› 30000 OK
60001 U. ‹RETURN› 60001 OK
-1 U. ‹RETURN› 65535 OK
-1000 U. ‹RETURN› 64536 OK
Die normale Ausgabe würde wie folgt aussehen:
-1 . ‹RETURN› - 1 OK
-1000 . ‹RETURN› -1000 OK

Operationen ohne Vorzeichen
U★ (u1 u2 --› ud)
Zwei vorzeichenlose Zahlen werden miteinander multipliziert. Das Ergebnis wird ebenfalls als vorzeichenlose doppelte Zahl abgelegt.

40000 44000 U★ D. ‹RETURN›
1760000000 OK
30000 34500 U★ D. ‹RETURN›
1035000000 OK
U/MOD (ud u1 --› u2 u3)

Eine doppelte Zahl wird durch eine einfache dividiert. Das Ergebnis und der Rest werden als einfache Zahl auf den Stack gelegt.

U/ (u1 u2 --› f)
Flag wird 1, wenn u1 kleiner als u2 ist.
100 1000 U/ . ‹RETURN› 1 OK
1000 100 U/ . ‹RETURN› 0 OK

FORTH bietet auch die Möglichkeit eine DO-LOOP-Schleife ohne Vorzeichen auszuführen.

DO (u1 u2 --› empty)
u1 gleich Endwert der Schleife und u2 gleich Anfangswert der Schleife.
LOOP (u --› empty)
u gleich Schrittweite der Schleife.

Beispiel:
```
Test 50000 0 DO
     CR
     I U.
     50001 /LOOP
TEST ‹RETURN›
0
5001
10002
15003
20004
25005
30006
35007
40008
45009 OK
```

Gleitkommazahlen
Bis jetzt haben wir viele Probleme ohne Gleitkommazahlen gelöst. Es ist also offensichtlich, daß man sehr oft auf die Rechnung in Gleitkommadarstellung verzichten kann. Gleichzeitig verzichtet man dabei auf die auftretenden Rundungsfehler, von denen Basic-Programmierer „ein Lied singen können". In den wenigen Fällen, wo man auf eine Berechnung in Gleitkommadarstellung nicht verzichten kann, wird man ein Gleitkommapaket, das zu vielen *FORTH-Versionen* erhältlich ist, hinzuladen.

Auf eine Darstellung dieser Befehlsworte kann in diesem Kurs nicht eingegangen werden, da es keine Standarts gibt und somit die Befehle stark von einander abweichen.

Diese sind jedoch in dem jeweiligen Manual ausführlich beschrieben.

Gesamtverzeichnis aller FORTH-Befehle: Wir haben nun eine Menge über die Sprache *FORTH* gelernt. Ziel dieses Kurses war es, dem Einsteiger einen Einblick und erste Programmierkenntnisse zu vermitteln. Für diejenigen unter Ihnen, die tiefer in die neue Sprache eindringen möchten, empfehlen wir am Ende noch einige interessante Literatur.

Zum Abschluß unseres *FORTH*-Kurses bringen wir noch einmal eine alphabetische Auflistung der wichtigsten Forth-Befehle und deren Aufgabe. Je nach *FORTH*-Version können mehr oder weniger Befehle implementiert sein. Dies ist aus dem jeweiligen Manual zu entnehmen.

! (n adr --> empty)
Speichere die 16-Bit-Zahl ›n‹ in die Adresse ›adr‹. Vergleichbar mit Poke in Basic, jedoch wird hier eine 16-Bit-Zahl gepokt.

!CSP (adr n --> empty)
Der aktuelle StackPointer wird in die User-Variable CSP gerettet.

(ud1 --> ud2)
Wandlung einer Zahlenstelle.
#> (ud --> adr n)

Ende der Formatierung. Auf dem Stapel bleibt die Adresse der Zeichenkette und die Anzahl der Zeichen.
#S (ud1 --> ud2)
Alle verbleibenden Stellen werden in ASCII Zeichen gewandelt.
' (empty --> adr)

Die Parameter-Feldadresse eines Befehlswortes oder einer Variablen wird auf dem Stack abgelegt.

< (empty --> empty)
Kommentarbeginn. Vergleichbar mit dem Basic-Befehl REM.
☆ (n1 n2 --> n3)
Das Produkt der Integer-Zahlen n1 und n2 wird auf dem Stack abgelegt.
☆/ (n1 n2 n3 --> n4)
Der Verhältnis-Faktor n4 wird aus der Formel n4 = n1☆n2/n3 errechnet. Das Produkt n1☆n2 wird doppelt so lang berechnet.
☆/MOD (n1 n2 n3 --> n4 n5)
Das gleiche wie der ☆/-Befehl. Jedoch wird auch hier der Rest n5 auf den Stack gelegt.
+ (n1 n2 --> n3)
Plus-Befehl! Die Summe aus den Zahlen n1 und n2 wird auf den Stack gelegt.
+! (n adr --> empty)
Die Zahl in Speicherstelle 1 wird um 1 erhöht.
+- (n1 n2 --> n3)
Das Vorzeichen von n2 wird auf die Zahl n1 angewandt. Das Ergebnis n3 wird auf dem Stack abgelegt.
+LOOP (n1 --> empty)
Das Zahlenparameter der Schleife wird um den Wert n1 geändert. Der vergleichbare Basic-Befehl wäre NEXT in Verbindung mit STEP.
- (n1 n2 --> n3)
Minus-Befehl! Die Differenz aus den Zahlen n1 und n2 wird auf dem Stack abgelegt.
--> (empty --> empty)
Dieser Befehl wird zum Verbinden mehrerer Screens genutzt. Er wird in der Regel auf die letzte Zeile des Screens gesetzt und bewirkt dadurch, daß der Compiler mit dem nächsten Screen fortfährt.
-DUP (n1 --> n1) falls n1 = 0
(n1 --> n1 n1) falls n1 <> 0
Die Zahl n1 wird mit 0 verglichen. Ist diese ungleich, wird sie dupliziert.
-TRAILING (adr n1 --> adr n2)
Ändert für die Ausgabe von Text die Parameter. Das führende Leerzeichen wird übersprungen.
. (n --> empty)
Die 16-Bit-Zahl wird auf dem Schirm ausgegeben. Beachtet wird die gewählte Zahlen-Basis.
." Beispiel: ." xxxxx"
Der Text xxxxx wird auf dem Schirm ausgegeben. Der vergleichbare Befehl wäre PRINT.
.R (n1 n2 --> empty)
Die Zahl n1 wird rechtsbündig in einem Fenster der Breite n2 ausgegeben. Der vergleichbare Basic-Befehl PRINT-USING.
/ (n1 n2 --> n3)
Divisions-Befehl! Der Quotient aus den Zahlen n1/n2 wird auf dem Stack abgelegt.
/MOD (n1 n2 --> n3 n4)
Das gleiche wie /-Befehl. Jedoch wird auch hier der Rest n3 auf dem Stack abgelegt.
Da 2diese Zahlen in *Forth* häu-

fig benutzt werden, sind diese Zahlen auch als Befehl definiert.

0< (n --> Flag)
Vergleichsbefehl! Ist die Zahl kleiner als 0, also im negativen Bereich, so wird das Flag auf 1 ansonsten auf 0 gesetzt.

0= (n --> Flag)
Vergleichsbefehl! Ist die Zahl gleich 0 so wird das Flag auf 1 ansonsten auf 0 gesetzt.

: (empty --> empty)
Beginn der Definition eines neuen Befehles.

; (empty --> empty)
Dies ist das Gegenstück zum >":"< Befehl. Das Ende einer Definition wird gekennzeichnet.

;S (empty --> empty)
Ende der Compilierung. Findet der *FORTH*-Compiler diesen Befehl vor, so beendet er den Compiler-Vorgang rechtzeitig.

< (n1 n2 --> Flag)
Vergleichsbefehl! Ist die Zahl n1 kleiner als n2, wird das Flag auf 1 ansonsten auf 0 gesetzt.

>R (n1 --> empty)
Die Zahl n wird zum Return-Stack gebracht. Mit diesem Befehl sollte sehr vorsichtig umgegangen werden, da es sonst leicht zum Absturz des Systems führen kann.

? (adr --> empty)
Der Inhalt der Adresse adr wird auf dem Schirm ausgegeben.

?TERMINAL (empty --> f)
Die Tastatur wird abgefragt. Ist eine Taste gedrückt, so wird das Flag auf 1 ansonsten auf 0 gesetzt.

□ (adr --> n)
Der Inhalt der Adresse adr wird auf den Stack gelegt. Der vergleichbare Basic-Befehl wäre PEEK, jedoch wird hier eine 16 Bit-Zahl gelesen.

ABS (n --> u)
Der Absolutwert von n wird auf dem Stack abgelegt. Mit anderen Worten, das Vorzeichen wird abgeschnitten.

ALLOT (n --> empty)
)n(Bytes Platz wird im Wörterbuch reserviert. Der Dictionary-Pointer DP wird also um n Bytes erhöht.

AND (n1 n2 --> n3)
Die Zahlen n1 und n2 werden logisch UND-verknüpft. Zu beachten ist, daß diese Verknüpfung bitweise vorgenommen wird.

BASE (n --> n adr)
Diese Variable enthält die gegenwärtig vereinbarte Zahlenbasis.

BLOCK (n --> adr)
Die Speicheradresse welche den Block n enthält wird ermittelt und auf den Stack gelegt.

C! (b adr --> empty)
Die-8 Bit-Zahl b wird in Adresse adr übertragen. Der vergleichbare Basic-Befehl wäre POKE.

C, (b --> empty)
Die-8 Bit-Zahl b wird in den nächsten freien Speicherplatz gebracht. Der Dictionary-Pointer wird um 1 erhöht.

C□ (adr --> b)
Die Zahl in Adresse adr wird als 8-Bit-Zahl auf dem Stack abgelegt. Das High-Byte wird also auf 0 gesetzt. Der vergleichbare Basic-Befehl wäre PEEK.

CFA (pfa --> cfa)
Die Parameterfeldadresse wird in die Codefeldadressé umgerechnet.

COMPILE
Das folgende Wort wird während der Laufzeit in das Wörterbuch eingetragen.

CMOVE (adr adr2 n --> empty)
Verschiebt n Byte von Adresse adr1 nach Adresse adr2.

CONSTANT (n --> empty)
Eine Konstante mit dem Wert n wird angelegt. Programme, welche zahlreiche Konstanten enthalten, sind in der Regel etwas schneller.

COPY (n1 n2 --> empty)
Das Textfeld n1 wird nach Textfeld n2 übertragen.

CONTEXT (empty --> adr)
Diese User-Variable legt die Anfangsadresse des aktuellen Wörterbuches auf den Stack.

COUNT(adr1 adr2 n)
Auf den Stapel wird die Länge und die Anfangsadresse eines Textes abgelegt.

CR
(empty --> empty)
Dieser Befehl bewirkt einen Zeilenvorschub mit Wagenrücklauf.

CREATE (empty --> empty)
Ist eines der wichtigsten Definitionsworte des FORTH-Kernes. Ein Wort wird in das Wörterbuch eingetragen und die Codefeldadresse wird Parameterfeldadresse des Eintrags.

CURRENT (empty --> adr)
Diese Variable enthält die Adresse des Wörterbuches, in die nachträgliche Definitionen eingetragen werden.

D+ (d1 d2 --> d3)
Zwei doppelt genaue Zahlen werden addiert und das Ergebnis auf dem Stack abgelegt.

D– (d1 d2 --> d3)
Die Differenz aus den beiden doppelt genauen Zahlen wird auf dem Stack abgelegt.

D. (d --> empty)
Eine doppelt genaue Zahl wird auf dem Schirm ausgegeben.

D.R (d n --> empty)
Die doppelt genaue Zahl d wird rechtsbündig in einem n breiten Fenster ausgegeben.

D0= (d --> f)
Vergleichsbefehl! Wenn die doppelt genaue Zahl gleich 0 ist, wird das Flag auf 1 gesetzt. Ansonsten auf 0.

D< (d1 d2 --> f)
Vergleichsbefehl! Wenn die doppelt genaue Zahl d2 kleiner als d1 ist, wird das Flag auf 1, ansonsten auf 0 gesetzt.

DECIMAL (empty --> empty)
Die dezimale Betriebsart wird aktiviert.

DO (n1 n2 --> empty)
Eine Schleife wird eröffnet mit dem Anfangsparameter n2 und dem Endparameter n1. Der vergleichbare Basic-Befehl wäre FOR.

DOES> (empty --> empty)
Dieses Wort wird in Definitionsworten verwendet, es beendet den Teil für die Compile-Anweisungen und zeigt den Beginn der Laufzeitanweisungen an.

DROP (n --> empty)
Die oberste Zahl des Stacks wird gelöscht.

DUP (n1 --> n1 n1)
Die oberste Zahl des Stack's wird dupliziert.

EMIT (c --> empty)
Die oberste Zahl des Stack's wird als ASCII-Zeichen auf dem Schirm ausgegeben.

ERASE (adr n --> empty)
Der Speicher wird ab Adresse adr mit n Bytes des Wertes 0 gefüllt.

EXECUTE (adr --> empty)
Ein Programm, welches bei der Adresse adr beginnt, wird zur Ausführung gebracht. Die Adresse entspricht der Codefeldadresse.

EXPECT (adr n --> empty)
Eine Anzahl von n Zeichen können an Adresse adr eingegeben werden.

FENCE (empty --> adr)
Diese Variable enthält die Adresse des ersten ungeschützten Wörterbucheintrages.

FILL (adr n c --> empty)
Der Speicher ab Adresse adr wird mit n Bytes des Wertes c gefüllt.

FLUSH (empty --> empty)
Das zuletzt bearbeitete Textfenster wird abgespeichert.

FORGET cccc (empty --> empty)
Alle Definitionen ab dem Wort cccc werden gelöscht.

FORTH (empty --> empty)
Das Wörterbuch mit der Bezeichnung *FORTH* wird aktiviert.

HERE (empty --> adr)
Dieser Befehl bringt die Adresse des nächsten freien Speicherplatzes auf den Stack.

HEX (empty --> empty)

Die Hexadezimale Betriebsart wird aktiviert.
HOLD (c --> empty)
Dieser Befehl erlaubt das Einfügen eines ASCII-Zeichens in eine Zeichenkette.
I (empty --> n)
Dieser Befehl wird innerhalb einer Schleife genutzt, um das aktuelle Zahlenparameter auf den Stack zu bringen.
IMMEDIATE empty --> empty)
Eine Definition wird als unmittelbar erklärt und sofort ausgeführt.
KEY (empty --> c)
Es wird ein Zeichen von der Tastatur eingelesen und als ASCII-Wert auf den Stack gelegt.
LATEST (empty --> adr)
Diese Variable enthält die Namensfeldadresse der letzten Definition.
LEAVE (empty --> empty)
Durch diesen Befehl kann eine DO-...LOOP Schleife verlassen werden.
LFA (pfa --> lfa)
Die Parameterfeldadresse wird in die Verbindungsfeldadresse umgerechnet.
LIST (n --> empty)
Das Textfenster n wird auf dem Schirm ausgelistet.
LITERIAL (n --> empty)
Bei der Compilierung von n wird vom Stack in die Wortdefinition eingefügt.
LORD (n --> empty)
Der Compiler beginnt bei Screen n zu compilieren.
M☆ (n1 n2 --> d1)
Die Integer-Zahlen n1 und n2 werden miteinander multipliziert und das Ergebnis als 32 Bit-Zahl auf den Stack gelegt.
M/ (d n1 --> n2)
Die doppelt genaue Zahl d wird durch Zahl n1 definiert und das Ergebnis auf den Stack gelegt.
MAX (n1 n2 --> n)
Die größere der beiden obersten Zahlen bleibt auf dem Stack.
MESSAGE (n --> empty)
Die Meldung n wird auf dem Schirm ausgegeben. Dieser Befehl wird vom *FORTH*-System zum Ausgeben der Fehlermeldungen verwendet.
MIN (n1 n2 --> n)
Die kleinere der beiden obersten Zahlen bleibt auf dem Stack.
MINUS (n --> -n)
Die oberste Zahl des Stacks wird negativ.
MOD (n1 n2 --> n)
Der Rest der Division n1 durch n2 wird auf dem Stack abgelegt.
NFA (pfa --> nfa)
Die Parameterfeldadresse wird in die Namensfeldadresse umgerechnet.
NUMMER (adr --> d)
Eine Zeichenkette, die bei Adresse adr+1 beginnt, wird in eine doppelt lange Zahl gewandelt. Die Adresse adr zeigt dabei auf die Länge der Zeichenkette.
OR (n1 n2 --> n)
Die Zahlen n1 und n2 werden miteinander durch OR verknüpft.
OVER (n1 n2 --> n1)
Die vorletzte Zahl des Stacks wird dupliziert und auf den Stack gelegt.
PAD (empty --> adr)
Die Adresse des Speichers namens PAD wird auf den Stack gelegt.
PFA (nfa --> pfa)
Die Namensfeldadresse wird in die Parameterfeldadresse umgerechnet.
QUIT (empty --> empty)
Die Laufzeit wird beendet ohne die Ausgabe von OK.
R (empty --> n)
Die oberste Zahl des RETURN-Stacks wird auf den Rechen-Stack gebracht, ohne sie auf dem RETURN-Stack zu zerstören.
R> (empty --> n)
Die oberste Zahl des RETURN-Stacks wird auf den Rechen-Stack geholt.
ROT (n1 n2 n3 --> n2 n3 n1)
Die obersten 3 Zahlen des Stacks werden um eine Stelle rotiert.
S0 (emply --> adr)
Diese Variable enthält die Adresse des Stackendes.
SCR (empty --> adr)
In dieser Variablen wird der zuletzt bearbeitete Screen festgehalten.
SP□ (empty --> adr)
Dieser Befehl legt die Adresse des augenblicklichen TOS auf den Stack. Der Befehl selbst wird von dieser Berechnung ausgeschlossen.
SPACE (empty --> empty)
Ein SPACE-Zeichen (dezimal 32) wird auf dem Schirm ausgegeben.
SPACES (n --> empty)
)n(SPACE-Zeichen werden auf dem Schirm ausgegeben.
SWAP (n1 n2 --> n2 n1)
Die beiden obersten Zahlen des Stacks werden vertauscht.
TIP (empty --> adr)
Diese Variable enthält die Startadresse des Terminal-Input-Puffers.
TYPE (adr n --> empty)
Eine Zeichenkette, deren Länge n ist und ab adr gespeichert wird, erscheint auf dem Schirm.
U☆ (u1 u2 --> ud)
Die beiden vorzeichenlosen 16 Bit-Zahlen werden miteinander multipliziert und das Ergebnis als doppelt genaue Zahl auf den Stack abgelegt.
U/ (ud u1 --> u2 u3)
Der Quotient der beiden Zahlen ud und u1 wird unter u3 und der Rest unter u2 auf den Stack gelegt.
UPDATE (empty --> empty)
Der zuletzt bearbeitete Screen wird als geändert markiert.
USER (name) (n --> empty)
Dieser Befehl trägt eine Variable in einen vom Benutzer definierten Bereich ein.
VARIABLE (name) (n --> empty)
Eine Variable (name) wird angelegt und mit dem Wert n geladen.
VOCABULARY (name) (empty --> empty)
Mit diesem Befehl läßt sich ein neues Wörterbuch eröffnen.
VLIST (empty --> empty)
Das eingeschaltete Wörterbuch wird auf dem Schirm ausgelistet.
WORD (c --> empty)
Eine Zeichenfolge wird aus dem Eingabespeicher (TIB) bis zum Begrenzungszeichen c übernommen.
XOR (n1 n2 --> n)
Die Zahlen n1 und n2 werden EXCLUSIV-ODER verknüpft und das Ergebnis auf dem Stack abgelegt.
[(empty --> empty)
Dieses Zeichen schaltet den Compiler aus.
] (empty --> empty)
Dieses Zeichen schaltet den Compiler ein.

Weiterführende Literatur:

STARING FORTH; Leo Brodie; englisch
Dieses Werk ist sicher eines der besten FORTH-Bücher, die es auf dem Markt gibt. Mit viel Humor und witzigen Skizzen wird man in die etwas schwierige Sprache Forths eingeführt. Leider gibt es bis jetzt noch keine deutsche Übersetzung.

FORTH HANDBUCH; E. Flögel; deutsch
Ein Buch, das zu den wenigen deutschen Büchern zählt, welche sich mit Forth beschäftigen. Der Leser wird Schritt für Schritt in den umfangreichen Wortschatz von FORTH eingeführt, dabei wird in erster Linie der FIG-Forth Standard besprochen. Einige kurze Programme runden dieses Werk ab.

DIE PROGRAMMIERSPRACHE FORTH; Ronald Zech; deutsch
In diesem Buch wird die Programmiersprache FORTH streng methodisch und umfassend beschrieben. Darüber hinaus wird Fundamentales erläutert, auf das die Sprache aufgebaut ist.
Da dieses Werk leider mit sehr vielen Fachwörtern umgeht, ist es nicht dem Anfänger zu empfehlen. Frank Brall

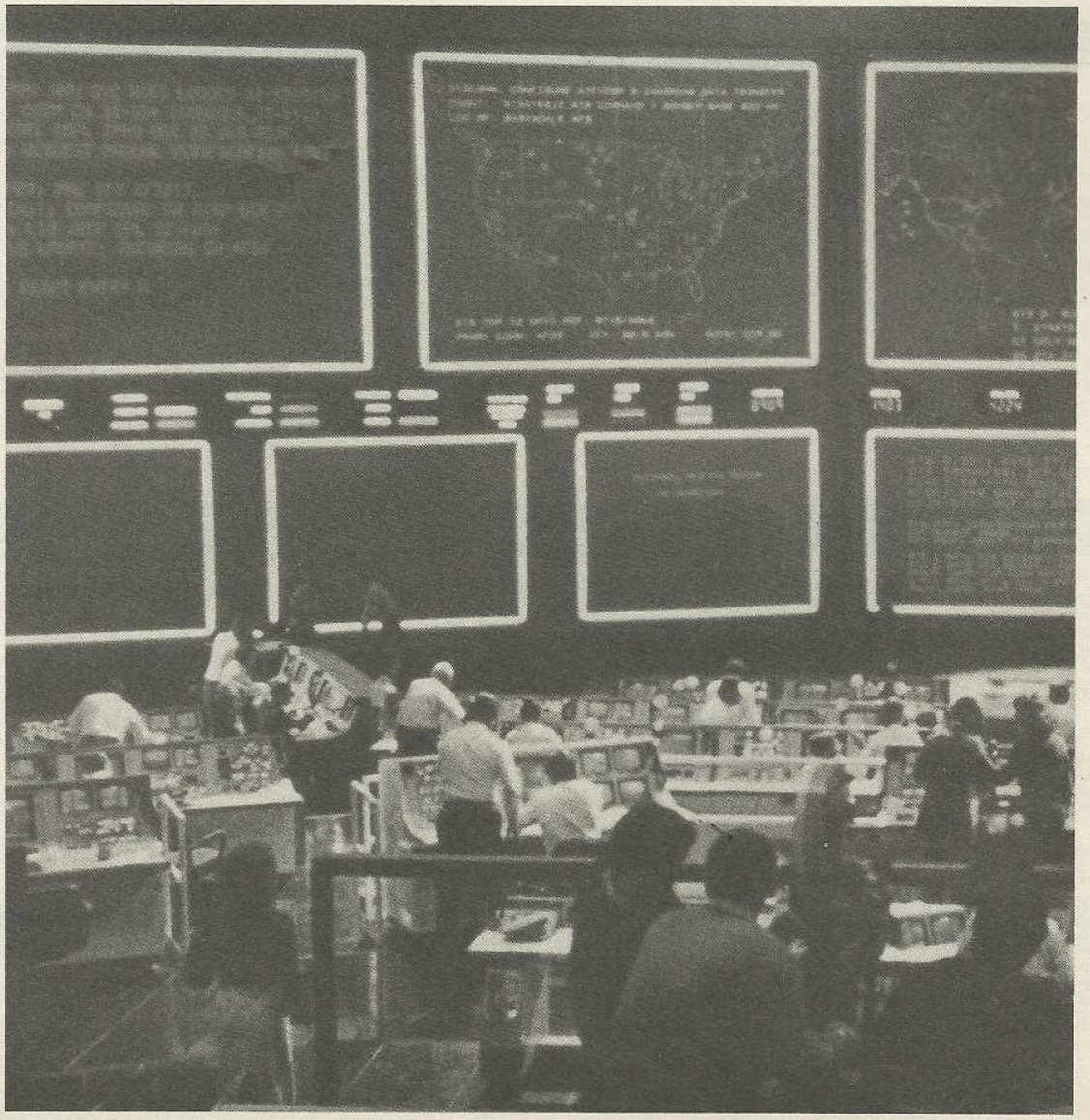
Aktuelles

Kurz notiert

Utopie oder Wirklichkeit?

Der amerikanische Film »Ware Games« beschreibt, wie ein Oberschüler aus Zufall den Code des geheimen Rechenzentrums der US-Luftwaffe knackt. Fast löst er mit seinen Spielereien den 3. Weltkrieg aus.
Daß dieser Film mittlerweile nicht nur reine Utopie ist, läßt sich aus mehreren amerikanischen Beispielen erkennen. Hier bauen sich Schüler automatische Anrufanlagen, die ununterbrochen verschiedene Nummern duchwählen bis eine andere Computeranlage erreicht wird. Dieser Code-Teil wird dann vom Computer automatisch gespeichert.
Auf diese Weise gelang es einem 14-jährigen »Hacker« aus den Staaten in die Datenbank des US-Verteidigungsministeriums einzudringen. Die Brüder Greg und Gary Knutson fanden mit ihrem Computer Einlaß in das geheime Datennetz der US-Weltraumbehörde NASA.

Wer ein Rechenzentrum anzapfen will muß schon ein Profi sein. So lernen in den Staaten die Kinder zum Beispiel schon in der Grundschule den Umgang mit den Computern und mancherorts wurden ganze Computer-Feriencamps aufgebaut.
In der BRD gibt es etwa 300 000 Heimcomputer, in den Staaten sind es über 1 600 000. Die US-Jugendlichen haben in der Regel das Wissen zum Umbau ihrer Computer, wodurch Sie dann in Rechenzentren eindringen können. Hierzu kommt noch, daß in den Staaten mehr öffentliche Rechenzentren zur Verfügung stehen, bei denen die Sicherheitsbestimmungen vernachlässigt werden. Es ist leichter an Telefonnummern heran zu kommen. Bei uns sind die »Hacker« noch auf Informationen von Mitarbeitern der Rechenzentren angewiesen. Jedoch haben sich in einigen Großstädten schon Schwarz-Märkte für diese Informationen gebildet.



Drei-Kronen-Spiel

Für den TI-99/4A in Extended Basic

Das Drei-Kronen-Spiel ist einem Geldspielautomaten nachempfunden, wie er in jeder Gaststätte hängt.

Eingabe des Geldbetrages:
Nachdem der Computer den Bildschirm aufgebaut hat, erwartet er die Eingabe eines Geldbetrages. Dieser muß mindestens 0,30 bzw. darf höchstens 99,9 DM sein. Hierbei ist das genaue Setzen des Punktes zu beachten.
Beispiel:
Eingabe Eins Punkt Zwei Null Bedeutung: 1,20 DM
Eingabe Eins Zwei Punkt Null Bedeutung: 12,00 DM,
hierbei wird die letzte Stelle ausgespart.
Jeder Spieldurchgang kostet 0,30 DM und wird von dem Geldspeicher automatisch abgezogen.

Start und Stop der Walzen:
Sollten auf den ersten beiden Walzen zwei Geldbeträge erscheinen, die Euch nicht gefallen, so könnt Ihr diese beim Aufleuchten des Wortes »START« erneut in Gang setzen. Um die mittlere Walze zu stoppen, braucht man nicht zu warten bis der Computer es tut, sondern kann dies beim Aufleuchten des Wortes »STOP« durch drücken einer Taste erreichen.

Gewinne:
Hier in tabellarischer Form die Gewinnmöglichkeiten.

	Linke Seite	Mitte	Rechte Seite	Gewinn
1.	0.40	0.40	0.40	0,40 DM
		Krone		
2.	0.60	0.60	0.60	0,60 DM
		Krone		
3.	0.80	0.80	0.80	0,80 DM
		Krone		
4.	1.20	1.20	1.20	1,20 DM
		Krone		
5.	1.60	1.60	1.60	1,60 DM
		Krone		
6.	1.80	1.80	1.80	1,80 DM
		Krone		
7.	2.40	2.40	2.40	2,40 DM
		Krone		
8.	3.00	3.00	3.00	3,00 DM
		Krone		
9.	Wappen	Wappen	Wappen	3,00 DM
				+ 5 Sonderspiele
10.	Wappen	Krone	Wappen	3,00 DM
				+10 Sonderspiele
11.	Krone	Krone	Krone	3,00 DM
				+20 Sonderspiele

Gewinnbalken:
Auf der rechten Seite des Bildschirmes sind 11 rote Balken, die zum Erreichen der Superserie benötigt werden. Sollten alle Balken weiß sein erklingt eine Tonfolge und Ihr habt 50 Superspiele gewonnen. Dieses sogenannte Aufleuchten der Balken wird erreicht, in dem Ihr mindestens einmal jeden der 11 oben aufgeführten Gewinne erzielt.

Beispiel:
Bei einem Gewinn von 0.40 0.40 bzw. Krone 0.40 wird der unterste Balken weiß; bei 0.60 0.60 bzw. Krone 0.60 wird der zweite Balken von unten weiß usw.
Wie Ihr seht, ist es nicht einfach an die Superserie zu kommen.

Gewinn in der Serie:
Jeder Gewinn in der Serie beträgt 3,00 DM.
Gewonnen wird bei:
a) drei gleichen Gewinnbeträgen
b) der Krone in der Mitte
c) dem Wappen in der Mitte
d) von 0,40 DM in der Mitte

RISIKO-Taste:
Dieses Risiko hat den Sinn die Spannung bei einem Gewinn zu erhöhen. Es erscheint im unteren Teil des Bildschirmes das Wort »RISIKO« und ein Farbband, das in schneller Folge die Farbe von rot nach weiß wechselt. Die Bedeutung der Farben sind:
rotes Band = Verlust des Gewinnes
weißes Band = Verdoppelung des Gewinnes
Es liegt in Eurer Hand, wenn Ihr versuchen wollt das weiße Farbband durch drücken einer Taste zu treffen. Für etwas ängstlichere Spieler bedeutet das Nichtdrücken einer Taste keinen Verlust des Gewinnes; sie bekommen halt nur den einfachen Geldbetrag.

Viel Erfolg!

Drei-Kronen-Spiel

```
5 CALL CLEAR :: CALL SCREEN(2)
10 DIM D$(10)
15 RESTORE 20 :: FOR A=1 TO 10 :: READ C$
20 DATA 0.40,0.60,0.80,1.20,1.60,1.80,2.40,3.00,4.00,5.00
25 D$(A)=C$ :: NEXT A
30 CALL CHAR(136,"C0C0C0C0C0C0C0C0",137,"0303030303030303",138,"0000000000000FFFF
")
35 CALL CHAR(139,"FFFF",140,"FFFFC0C0C0C0C0C0",141,"0303030303030FFFF")
40 CALL CHAR(142,"C0C0C0C0C0C0FFFF",143,"FFFF0303030303030303",128,"FFFFFFFFFFFF")
45 CALL CHAR(129,"FCFCFCFCFCFCFCFC",130,"0000FFFFFFFFFFFF",131,"3F3F3F3F3F3F3F3F
")
50 CALL CHAR(120,"FFFFFFFFFFFFFFFF",121,"FEFCF8F0E0C08000",122,"0103070F1F3F7FFF
")
55 CALL CHAR(123,"80C0E0F0F8FCFEFF",124,"7F3F1F0F07030100",112,"FFFFFFFFFFFFFFFF
")
60 CALL CHAR(113,"0000000000000FFFF",114,"C0C0C0C0C0C0C0C0",115,"FFFF")
65 CALL CHAR(104,"FFFFFFFFFFFFFFFF",125,"")
70 CALL CHAR(36,"20313B3F3B352E3D3F3F3F1F0F07030104 8CDCFCFCFCFC5CAC74ACD8F0E0C08
0")
75 CALL CHAR(40,"0101074141415577777F7F7F7F7F7F008080E0828282AAEEEEFEFEFEFEFEFE0
0")
80 CALL CHAR(92,"03070E1C3870E0C0000000000000000000000000000000003070E1C3870E0C
0")
85 CALL CHAR(96,"0000000000000000000000000000000C0E070381C0E0703000000000000000
0")
90 CALL CHAR(100,"0000000000000000000000000000000000000000000000003070E1C3870E0
C0")
95 CALL CHAR(108,"00000000000000C0E070381C0E0703000000000000000000000000000000
00")
100 CALL CHAR(116,"03070E1C3870E0C0000000000000000000000000000000000000000000
000")
105 CALL CHAR(132,"00000000000000C0E070381C0E0703C0E070381C0E0703000000000000
000")
110 FOR A=4 TO 20 :: CALL HCHAR(A,1,120,26):: NEXT A
115 FOR A=1 TO 24 :: CALL HCHAR(A,27,125,6):: NEXT A
120 CALL HCHAR(3,1,113,26):: CALL HCHAR(21,1,115,26):: CALL VCHAR(1,27,136,24)
125 CALL HCHAR(1,1,115,26):: CALL HCHAR(24,1,113,26)
130 CALL HCHAR(1,27,139,6):: CALL HCHAR(24,27,138,6):: CALL HCHAR(1,27,140,1)::
CALL HCHAR(24,27,142,1)
135 FOR A=2 TO 22 STEP 2 :: CALL HCHAR(A,29,104,3):: NEXT A
140 RESTORE 145 :: FOR A=1 TO 6 :: READ B1,B2,B3
145 DATA 2,6,3,2,16,6,22,6,5,23,6,5,22,15,7,23,15,7
150 CALL HCHAR(B1,B2,125,B3):: NEXT A
155 RESTORE 160 :: FOR A=1 TO 15 :: READ B1,B2
160 DATA 6,6,7,6,8,6,6,18,7,18,8,18,11,12,12,12,13,12,16,6,17,6,18,6,16,18,17,18
,18,18
165 CALL HCHAR(B1,B2,125,4):: NEXT A
170 CALL HCHAR(1,6,139,3):: CALL HCHAR(1,16,139,6):: CALL HCHAR(21,6,139,5):: CA
LL HCHAR(21,15,139,7)
175 CALL HCHAR(3,6,138,3):: CALL HCHAR(3,16,138,6):: CALL HCHAR(24,6,138,5):: CA
LL HCHAR(24,15,138,7)
180 CALL HCHAR(1,5,140,1):: CALL HCHAR(1,9,143,1):: CALL HCHAR(3,5,142,1):: CALL
 HCHAR(3,9,141,1)
185 CALL HCHAR(1,15,140,1):: CALL HCHAR(1,22,143,1):: CALL HCHAR(3,15,142,1):: C
ALL HCHAR(3,22,141,1)
```

```
190 CALL HCHAR(21,5,140,1):: CALL HCHAR(21,11,143,1):: CALL HCHAR(24,5,142,1)::
CALL HCHAR(24,11,141,1)
195 CALL HCHAR(21,14,140,1):: CALL HCHAR(21,22,143,1):: CALL HCHAR(24,14,142,1):
: CALL HCHAR(24,22,141,1)
200 RESTORE 205 :: FOR A=1 TO 5 :: READ B1,B2
205 DATA 5.6.5.18.10.12.15.6.15.18
210 CALL HCHAR(B1,B2,130,4):: CALL HCHAR(B1+4,B2,128,4):: NEXT A
215 RESTORE 220 :: FOR A=1 TO 5 :: READ B1,B2
220 DATA 6,5,6,17,11,11,16,5,16,17
225 CALL VCHAR(B1,B2,131,3):: CALL VCHAR(B1,B2+5,129,3):: NEXT A
230 CALL HCHAR(2,5,136,1):: CALL HCHAR(2,15,136,1):: CALL VCHAR(22,5,136,2):: CA
LL VCHAR(22,14,136,2)
235 CALL HCHAR(2,9,137,1):: CALL HCHAR(2,22,137,1):: CALL VCHAR(22,11,137,2):: C
ALL VCHAR(22,22,137,2)
240 RESTORE 245 :: FOR A=1 TO 5 :: READ B1,B2
245 DATA 5,5,5,17,10,11,15,5,15,17
250 CALL HCHAR(B1,B2,121,1):: CALL HCHAR(B1+4,B2,123,1):: CALL HCHAR(B1,B2+5,124
,1):: CALL HCHAR(B1+4,B2+5,122,1):: NEXT A
255 CALL COLOR(14,16,6,13,6,2,12,5,6,11,16,2,10,7,16):: CALL MAGNIFY(3)
260 FOR A=2 TO 9 :: CALL COLOR(A,2,6):: NEXT A
265 CALL SPRITE(#1,100,2,25,25,#2,100,2,25,121,#3,100,2,105,25)
270 CALL SPRITE(#4,108,2,25,73,#5,108,2,25,169,#6,108,2,105,169)
275 CALL SPRITE(#7,96,2,65,25,#8,96,2,145,25,#9,96,2,145,121)
280 CALL SPRITE(#10,116,2,65,169,#11,116,2,145,73,#12,116,2,145,169)
285 CALL SPRITE(#13,92,2,65,73,#14,92,2,105,121,#15,132,2,65,121,#16,132,2,105,7
3)
290 S$="000" :: G$="000.00"
295 DISPLAY AT(2,4)SIZE(3):S$ :: DISPLAY AT(2,14)SIZE(6):G$
300 DISPLAY AT(7,10)SIZE(4):"DREI" :: DISPLAY AT(12,1)SIZE(6):"KRONEN" :: DISPLA
Y AT(12,17)SIZE(5):"SPIEL"
305 RESTORE 310 :: FOR A=2 TO 22 STEP 2 :: READ B1
310 DATA 83,85,80,69,82,47,83,69,82,73,69
315 DISPLAY AT(A,26)SIZE(1):CHR$(B1):: NEXT A
320 FOR A=1 TO 50 :: NEXT A
325 DISPLAY AT(22,13)SIZE(7):"BETRAG?"
330 ACCEPT AT(23,14)VALIDATE(DIGIT,".")SIZE(-4):M1
335 IF M1<0.3 XOR M1>99.9 THEN 340 ELSE 345
340 CALL HCHAR(23,15,125,7):: GOTO 330
345 M=M+M1 :: GOSUB 1065
350 CALL HCHAR(22,15,125,7):: CALL HCHAR(23,15,125,7)
354 FOR A=1 TO 100 :: NEXT A
355 REM ** LINKEN WALZEN **
360 B4=0 :: M=M-.3 :: GOSUB 1065
365 B4=B4+1 :: RANDOMIZE
370 U=INT(RND*7+3)
375 X(1)=U :: H=17 :: V=4
380 IF U=9 THEN CALL SPRITE(#20,40,2,125,49)ELSE GOSUB 570
385 CALL WALZE(U,U1)
390 X(4)=U1
395 IF U1=10 THEN CALL SPRITE(#21,36,12,45,49)ELSE DISPLAY AT(7,4)SIZE(4):D$(U1)
400 IF B4=2 THEN 430 :: DISPLAY AT(22,4)SIZE(5):"START"
405 FOR A=1 TO 50 :: CALL KEY(0,K,S):: IF S<>0 THEN 420
410 NEXT A
415 GOTO 430
420 CALL HCHAR(7,6,125,4):: CALL HCHAR(17,6,125,4):: CALL DELSPRITE(#20,#21)
425 CALL HCHAR(22,6,125,5):: GOTO 365
430 REM ** RECHTEN WALZEN **
435 CALL HCHAR(22,6,125,5)
440 FOR A=1 TO 120 :: NEXT A
445 RANDOMIZE
450 U=INT(RND*7+3)
```

```
455 X(2)=U :: H=17 :: V=16
460 IF U=9 THEN CALL SPRITE(#23,40,2,125,145)ELSE GOSUB 570
465 CALL WALZE(U,U1)
470 X(5)=U1
475 IF U1=10 THEN CALL SPRITE(#24,36,12,45,145)ELSE DISPLAY AT(7,16)SIZE(4):D$(U
1)
480 REM ** MITTLERE WALZE **
485 FOR A=1 TO 120 :: NEXT A
490 DISPLAY AT(22,4)SIZE(4):"STOP"
495 B5=0
500 RANDOMIZE
505 B5=INT(RND*10+1)
510 FOR A=1 TO 50
515 B5=B5+1
520 CALL KEY(0,K,S)
525 IF S<>0 THEN 550
530 IF B5=10 THEN B5=0
535 NEXT A
540 RANDOMIZE
545 B5=INT(RND*10+1)
550 IF B5>8 THEN 560
555 X(3)=B5 :: DISPLAY AT(12,10)SIZE(4):D$(B5):: GOTO 575
560 IF B5=9 THEN CALL SPRITE(#25,40,2,85,97)ELSE CALL SPRITE(#25,36,12,85,97)
565 CALL HCHAR(22,6,125,5):: X(3)=B5 :: GOTO 575
570 DISPLAY AT(H,V)SIZE(4):D$(U):: RETURN
575 IF J>0 THEN 950 ELSE DISPLAY AT(2,4)SIZE(3):"000"
580 REM ** GEWINNABFRAGE **
585 IF X(1)=9 AND X(2)=9 AND X(3)=9 THEN 865
590 IF X(4)=10 AND X(5)=10 AND X(3)=10 THEN 805
595 IF X(3)=9 AND X(4)=10 AND X(5)=10 THEN 835
600 IF ((X(1)=X(2)XOR X(1)=X(5))AND(X(1)=X(3)XOR X(3)=9))THEN 620 ELSE IF ((X(4)
=X(5)XOR X(4)=X(2))AND(X(4)=X(3)XOR X(3)=9))THEN 620
605 IF X(3)=9 THEN 925
610 GOTO 1180
615 REM ** RISIKOLAUF **
620 DISPLAY AT(22,13)SIZE(6):"RISIKO"
625 FOR A=1 TO 50
630 CALL SOUND(-100,880,6)
635 CALL HCHAR(23,15,112,6)
640 CALL KEY(0,K,S):: IF S=0 THEN 645 ELSE 710
645 CALL HCHAR(23,15,104,6)
650 CALL KEY(0,K,S):: IF S<>0 THEN 775
655 NEXT A
660 IF (X(1)=X(2)XOR X(1)=X(5))AND(X(1)=X(3)XOR X(3)=9)THEN 665 ELSE 670
665 L=1 :: GOTO 675
670 L=4
675 GOSUB 1225
680 ON X(L)GOSUB 1095,1100,1105,1110,1115,1120,1125,1130
685 GOSUB 1065
690 GOSUB 1010
695 L=0 :: CALL HCHAR(22,15,125,7):: CALL HCHAR(23,15,125,7)
700 GOTO 1140
705 REM ** RISIKO-GEWINN **
710 CALL SOUND(1500,262,3,330,3,392,3)
715 CALL HCHAR(23,15,125,6):: CALL HCHAR(22,15,125,7)
720 IF (X(1)=X(2)XOR X(1)=X(5))AND(X(1)=X(3)XOR X(3)=9)THEN 730
725 IF (X(4)=X(5)XOR X(4)=X(2))AND(X(4)=X(3)XOR X(3)=9)THEN 735
730 L=1 :: GOTO 740
735 L=4
740 FOR A=1 TO 8
745 IF X(L)=A THEN M=M+(VAL(D$(A)))
```

```
750 NEXT A
755 ON X(L)GOSUB 1095,1100,1105,1110,1115,1120,1125,1130
760 GOSUB 1065
765 GOSUB 1010
770 GOTO 1140
775 REM ** RISIKO-VERLUST **
780 CALL SOUND(-300,500,3)
785 CALL HCHAR(23,15,125,6)
790 CALL HCHAR(22,15,125,7)
795 FOR A=1 TO 100 :: NEXT A
800 GOTO 1140
805 REM ** SONDERSPIELE BEI 3 WAPPEN **
810 GOSUB 1280
815 J=J+5 :: GOSUB 900
820 CALL HCHAR(6,29,112,3):: GOSUB 1010
825 GOSUB 1065
830 GOTO 1140
835 REM ** SONDERSPIELE BEI 2 WAPPEN UND 1 KRONE **
840 GOSUB 1280
845 J=J+10 :: GOSUB 895
850 CALL HCHAR(4,29,112,3):: GOSUB 1010
855 GOSUB 1065
860 GOTO 1140
865 REM ** SONDERSPIELE BEI 3 KRONEN **
870 GOSUB 1280
875 J=J+20 :: GOSUB 895
880 CALL HCHAR(2,29,112,3):: GOSUB 1010
885 GOSUB 1065
890 GOTO 1140
895 REM ** SONDERSPIELSCHLEIFE **
900 M=M+3
905 IF J>=100 THEN 915 ELSE IF J>=10 THEN 920
910 DISPLAY AT(2,4)SIZE(2):"00" :: DISPLAY AT(2,6)SIZE(1):USING "#":J :: RETURN
915 DISPLAY AT(2,4)SIZE(3):USING "###":J :: RETURN
920 DISPLAY AT(2,4)SIZE(1):"0" :: DISPLAY AT(2,5)SIZE(2):USING "##":J :: RETURN
925 REM ** MITTLERE WALZE MIT KRONE **
930 GOSUB 1225
935 M=M+.3 :: GOSUB 1065
940 GOTO 1140
945 REM ** GEWINN INNERHALB DER SERIE **
950 J=J-1
955 IF X(1)=9 AND X(2)=9 AND X(3)=9 THEN 865
960 IF ((X(1)=X(2)XOR X(1)=X(5))AND(X(1)=X(3)XOR X(3)=9))XOR((X(4)=X(5)XOR X(4)=
X(2))AND(X(4)=X(3)XOR X(3)=9))THEN M=M+3
965 IF X(4)=10 AND X(5)=10 AND X(3)=10 THEN 805
970 IF X(3)=9 AND X(4)=10 AND X(5)=10 THEN 835
975 IF X(3)=9 XOR X(3)=10 XOR X(3)=1 THEN M=M+3 ELSE 985
980 GOSUB 1255
985 GOSUB 905
990 GOSUB 1065
995 GOSUB 1010
1000 GOTO 1140
1005 REM ** ABFRAGE AUF SUPERSERIE **
1010 FOR A=2 TO 22 :: CALL GCHAR(A,29,A2)
1015 IF A2=112 THEN A3=A3+1
1020 NEXT A
1025 IF A3=11 THEN 1035
1030 A3=0 :: RETURN
1035 REM ** SUPERSERIE **
1040 J=J+50 :: GOSUB 905
1045 A3=0
```

```
1050 FOR A=2 TO 22 STEP 2 :: CALL HCHAR(A,29,104,3):: NEXT A
1055 GOTO 1140
1060 REM ** AUSGABE DES GEWINNES **
1065 IF M>=100.0 THEN 1085 ELSE IF M>=10.0 THEN 1080 ELSE IF M>=1.0 THEN 1075
1070 DISPLAY AT(2,14)SIZE(3):"000" :: DISPLAY AT(2,17)SIZE(2):USING ".#":M :: RE
TURN
1075 DISPLAY AT(2,14)SIZE(2):"00" :: DISPLAY AT(2,16)SIZE(3):USING "#.#":M :: RE
TURN
1080 DISPLAY AT(2,14)SIZE(1):"0" :: DISPLAY AT(2,15)SIZE(4):USING "##.#":M :: RE
TURN
1085 DISPLAY AT(2,14)SIZE(5):USING "###.#":M :: RETURN
1090 REM ** GEWINNBALKEN **
1095 M=M+.4 :: CALL HCHAR(22,29,112,3):: RETURN
1100 M=M+.6 :: CALL HCHAR(20,29,112,3):: RETURN
1105 M=M+.8 :: CALL HCHAR(18,29,112,3):: RETURN
1110 M=M+1.2 :: CALL HCHAR(16,29,112,3):: RETURN
1115 M=M+1.6 :: CALL HCHAR(14,29,112,3):: RETURN
1120 M=M+1.8 :: CALL HCHAR(12,29,112,3):: RETURN
1125 M=M+2.4 :: CALL HCHAR(10,29,112,3):: RETURN
1130 M=M+3 :: CALL HCHAR(8,29,112,3):: RETURN
1135 REM ** LOESCHEN DES ALTEN SPIELS **
1140 CALL HCHAR(22,6,125,5)
1145 FOR A=1 TO 100 :: NEXT A
1150 RESTORE 1155 :: FOR A=1 TO 5 :: READ B1,B2
1155 DATA 7,6,7,18,12,12,17,6,17,18
1160 CALL HCHAR(B1,B2,125,4):: NEXT A
1165 FOR A=20 TO 25 :: CALL DELSPRITE(#A):: NEXT A
1170 IF M<.3 THEN 325 ELSE IF M>999.9 THEN 1175 ELSE 360
1175 END
1180 CALL SOUND(250,440,6)
1185 CALL SOUND(150,392,6)
1190 CALL SOUND(150,330,6)
1195 CALL SOUND(150,294,6)
1200 CALL SOUND(150,277,6)
1205 CALL SOUND(250,262,6)
1210 CALL SOUND(350,220,6)
1215 CALL SOUND(450,196,6)
1220 GOTO 1140
1225 CALL SOUND(-330,800,5)
1230 CALL SOUND(-330,988,5)
1235 CALL SOUND(-330,784,5)
1240 CALL SOUND(-330,392,5)
1245 CALL SOUND(-990,578,5)
1250 RETURN
1255 CALL SOUND(300,415,5)
1260 CALL SOUND(300,311,5)
1265 CALL SOUND(300,523,5)
1270 CALL SOUND(300,415,5)
1275 RETURN
1280 CALL SOUND(200,440,5)
1285 CALL SOUND(200,550,5)
1290 CALL SOUND(200,660,5)
1295 CALL SOUND(400,880,5)
1300 RETURN
1305 SUB WALZE(U,U1)
1310 F=U-2
1315 ON F GOTO 1320,1320,1325,1335,1340,1345,1355
1320 U1=1 :: GOTO 1365
1325 U1=INT(RND*3+1)
1330 IF U1=2 THEN 1325 ELSE 1365
1335 U1=INT(RND*2+2):: GOTO 1365
1340 U1=INT(RND*4+1):: GOTO 1365
1345 U1=INT(RND*10+1)
1350 IF U1=3 XOR U1=6 XOR U1=10 THEN 1365 ELSE 1345
1355 U1=INT(RND*3+8)
1360 IF U1=9 THEN 1355
1365 SUBEND
```

Fortsetzung

Zahlenputzen

Für den TI-99/4A in Extended Basic

Bei diesem Listing handelt es sich um ein Würfelspiel. Ihr müßt durch geschicktes Überlegen die Zahlen 1-9 mit Hilfe zweier Würfel ausputzen. Genauere Anweisungen findet Ihr im Programm selbst.

Zahlenputzen

```
100 CALL CLEAR :: CALL SCREEN(2)
110 CALL CHAR(136,"CCCC3333CCCC3333",120,"FFFFFFFFFFFFFFFF",128,"FFFFFFFFFFFFFFF
F",121,"",129,"")
120 FOR A=3 TO 8 :: CALL COLOR(A,15,2):: NEXT A
130 CALL HCHAR(6,9,42,16):: CALL HCHAR(10,9,42,16):: CALL VCHAR(6,9,42,5):: CALL
 VCHAR(6.24.42.5)
140 CALL HCHAR(15,3,128,28)
150 CALL HCHAR(4,7,136,20):: CALL HCHAR(12,7,136,20):: CALL VCHAR(4,7,136,9):: C
ALL VCHAR(4,26,136,9)
160 CALL COLOR(2,5,2,13,13,5,12,11,7)
170 CALL HCHAR(1,1,42,32):: CALL VCHAR(1,32,42,24)
180 FOR C=32 TO 1 STEP -1 :: CALL HCHAR(24,C,42,1):: NEXT C
190 FOR C=24 TO 1 STEP -1 :: CALL VCHAR(C,1,42,1):: NEXT C
200 CALL HCHAR(8,11,65,12)
210 FOR H=11 TO 22 :: READ ZE
220 FOR N=65 TO ZE :: CALL SOUND(-1,333,3):: CALL HCHAR(8,H,N):: NEXT N
230 CALL SOUND(100,300,3):: NEXT H
240 DATA 90,65,72,76,69,78,80,85,84,90,69,78
250 CALL COLOR(2,11,2)
260 FOR A=1 TO 15 :: CALL COLOR(14,7,11,2,5,2):: CALL HCHAR(15,3,129,28)
270 FOR B=1 TO 20 :: NEXT B :: CALL COLOR(14,11,7,2,11,2):: CALL HCHAR(15,3,120,
28)
280 FOR B=1 TO 20 :: NEXT B :: CALL COLOR(14,16,5,2,5,2):: CALL HCHAR(15,3,121,2
8)
290 FOR B=1 TO 20 :: NEXT B :: CALL COLOR(14,5,16,2,11,2):: CALL HCHAR(15,3,128,
28)
300 FOR B=1 TO 20 :: NEXT B
310 NEXT A
320 CALL SOUND(1500,262,3,330,3,392,3):: DISPLAY AT(18,8)SIZE(12):"ULRICH HOLLE"
 :: DISPLAY AT(20,8)SIZE(12):"AM ECKLAND 4"
330 DISPLAY AT(22,4)SIZE(21):"4330 MUELHEIM / RUHR"
340 FOR A=1 TO 800 :: NEXT A
350 CALL CHARSET :: CALL CLEAR
360 CALL SCREEN(15)
370 HS=0 :: DURCHGANG=0
380 DISPLAY AT(10,3)SIZE(23):"WILLST DU ANWEISUNGEN ?"
390 DISPLAY AT(12,11)SIZE(7):"( J/N )"
400 CALL KEY(0,K,S)
410 IF S=0 THEN 400
420 IF K=74 THEN 1960 ELSE IF K=78 THEN CALL CLEAR :: CALL SCREEN(2):: GOTO 430
ELSE 400
430 CALL CHAR(136,"8080808080808080")
440 CALL CHAR(137,"FF")
450 CALL CHAR(138,"0101010101010101")
460 CALL CHAR(139,"00000000000000FF")
470 CALL CHAR(140,"FF8080808080808080")
480 CALL CHAR(141,"FF0101010101010101")
490 CALL CHAR(96,"00000000000000010100000000000000000000000000008080000000000000
00")
500 CALL CHAR(142,"01010101010101FF")
510 CALL CHAR(100,"00000018180000000000000000000000000000000000000000000018180000
000")
520 CALL CHAR(143,"8080808080808080FF")
```

```
530 CALL CHAR(104,"0000001818000001010000000000000000000000000000808000001818000
000")
540 CALL CHAR(128,"FFFEFCF8F0E0C080")
550 CALL CHAR(108,"0000001818000000000001818000000000001818000000000001818000
000")
560 CALL CHAR(129,"FFFFFFFFFFFFFFFF")
570 CALL CHAR(112,"0000001818000001010000181800000000000018180000808000001818000
000")
580 CALL CHAR(130,"0103070F1F3F7FFF")
590 CALL CHAR(116,"0000001919000000000000191900000000000098980000000000009898000
000")
600 CALL HCHAR(18,8,137,4):: CALL HCHAR(18,21,137,4):: CALL HCHAR(21,8,139,4)::
CALL HCHAR(21,21,139,4)
610 CALL VCHAR(18,8,136,4):: CALL VCHAR(18,21,136,4):: CALL VCHAR(18,11,138,4)::
 CALL VCHAR(18,24,138,4)
620 CALL HCHAR(18,8,140,1):: CALL HCHAR(18,11,141,1):: CALL HCHAR(21,8,143,1)::
CALL HCHAR(21,11,142,1)
630 CALL HCHAR(18,21,140,1):: CALL HCHAR(18,24,141,1):: CALL HCHAR(21,21,143,1):
: CALL HCHAR(21,24,142,1)
640 CALL HCHAR(17,9,129,4):: CALL HCHAR(17,8,130,1):: CALL VCHAR(17,12,129,4)::
CALL HCHAR(21,12,128,1)
650 CALL HCHAR(17,22,129,4):: CALL HCHAR(17,21,130,1):: CALL VCHAR(17,25,129,4):
: CALL HCHAR(21,25,128,1)
660 CALL HCHAR(11,4,137,27):: CALL HCHAR(13,4,139,27)
670 RESTORE 680 :: FOR A=1 TO 9 :: READ A1
680 DATA 4,7,10,13,16,19,22,25,28
690 CALL HCHAR(11,A1,140,1):: NEXT A
700 RESTORE 710 :: FOR A=1 TO 9 :: READ A1
710 DATA 6,9,12,15,18,21,24,27,30
720 CALL HCHAR(11,A1,141,1):: NEXT A
730 RESTORE 740 :: FOR A=1 TO 9 :: READ A1
740 DATA 4,7,10,13,16,19,22,25,28
750 CALL HCHAR(12,A1,136,1):: CALL HCHAR(13,A1,143,1):: NEXT A
760 RESTORE 770 :: FOR A=1 TO 9 :: READ A1
770 DATA 6,9,12,15,18,21,24,27,30
780 CALL HCHAR(13,A1,142,1):: CALL HCHAR(12,A1,138,1):: NEXT A
790 CALL VCHAR(11,3,138,3):: CALL VCHAR(11,31,136,3):: CALL HCHAR(10,4,139,27)::
 CALL HCHAR(14,4,137,27)
800 CALL HCHAR(1,4,139,7):: CALL HCHAR(1,23,139,7):: CALL HCHAR(2,4,137,7):: CAL
L HCHAR(2,23,137,7):: CALL HCHAR(4,4,139,7)
810 CALL HCHAR(5,4,137,7):: CALL HCHAR(4,23,139,7):: CALL HCHAR(5,23,137,7):: CA
LL VCHAR(2,3,138,3):: CALL VCHAR(2,11,136,3)
820 CALL HCHAR(3,4,136,1):: CALL HCHAR(3,10,138,1):: CALL HCHAR(2,4,140,1):: CAL
L HCHAR(2,10,141,1):: CALL HCHAR(4,4,143,1)
830 CALL HCHAR(4,10,142,1):: CALL HCHAR(2,23,140,1):: CALL HCHAR(2,29,141,1):: C
ALL HCHAR(4,23,143,1):: CALL HCHAR(4,29,142,1)
840 CALL HCHAR(3,23,136,1):: CALL HCHAR(3,29,138,1):: CALL HCHAR(4,15,139,3):: C
ALL HCHAR(5,15,137,3):: CALL HCHAR(7,15,139,3)
850 CALL HCHAR(8,15,137,3):: CALL HCHAR(5,15,140,1):: CALL HCHAR(5,17,141,1):: C
ALL HCHAR(7,15,143,1):: CALL HCHAR(7,17,142,1)
860 CALL HCHAR(6,15,136,1):: CALL HCHAR(6,17,138,1):: CALL VCHAR(5,14,138,3):: C
ALL VCHAR(5,18,136,3)
870 CALL SCREEN(11):: CALL COLOR(14,2,11,13,2,11)
880 A1=48
890 FOR A=5 TO 29 STEP 3
900 A1=A1+1
910 CALL HCHAR(12,A,A1,1):: NEXT A
920 DISPLAY AT(6,3)SIZE(5):"SCORE" :: DISPLAY AT(6,20)SIZE(9):"HIGHSCORE" :: DIS
PLAY AT(9,10)SIZE(9):"BONUSZAHL"
930 DISPLAY AT(3,26)SIZE(1):USING "#":"0"
940 DISPLAY AT(3,7)SIZE(1):USING "#":"0"
```

```
950 PUNKTE=0 :: DURCHGANG=1
960 BONUS=4
970 DISPLAY AT(6,14)SIZE(1):USING "#":BONUS
980 FOR A=5 TO 29 STEP 3
990 CALL GCHAR(12,A,DY)
1000 IF DY=32 THEN SUM=SUM+1
1010 NEXT A
1020 IF SUM=9 THEN 1770 ELSE SUM=0
1030 FOR A=1 TO 250 :: NEXT A
1040 Y=92
1050 FOR A=1 TO 6
1060 Y=Y+4
1070 Z(A)=Y
1080 NEXT A
1090 RANDOMIZE :: CALL MAGNIFY(4)
1100 FOR A=1 TO 10
1110 Y1=INT(RND*8)+1
1120 IF Y1>6 THEN 1110
1130 Y2=INT(RND*8)+1
1140 IF Y2>6 THEN 1130
1150 CALL SPRITE(#1,Z(Y1),2,137,57,#2,Z(Y2),2,137,161)
1160 FOR A1=1 TO 40 :: NEXT A1
1170 NEXT A
1180 WERT=Y1+Y2
1190 IF Y1=Y2 THEN BONUS=BONUS+2
1200 IF BONUS>9 THEN BONUS=9
1210 DISPLAY AT(6,14)SIZE(1):USING "#":BONUS
1220 CALL KEY(0,K,S)
1230 IF S=0 THEN 1220
1240 IF K=32 THEN 1580
1250 IF K<49 XOR K>57 THEN 1220
1260 FOR A=5 TO 29 STEP 3
1270 CALL GCHAR(12,A,FF)
1280 IF K=FF THEN 1310
1290 NEXT A
1300 GOTO 1220
1310 K=K-48
1320 DIF=WERT-K
1330 IF DIF=0 THEN 1510
1340 IF DIF>9 XOR DIF<1 THEN 1350 ELSE 1360
1350 WERT=Y1+Y2 :: GOTO 1220
1360 DIF=DIF+48
1370 FOR A=5 TO 29 STEP 3
1380 CALL GCHAR(12,A,FG)
1390 IF FG=DIF THEN 1430
1400 NEXT A
1410 DIF=DIF-48 :: WERT=Y1+Y2 :: GOTO 1220
1420 DIF=DIF+48
1430 K=K+48
1440 FOR A=5 TO 29 STEP 3
1450 CALL GCHAR(12,A,DE)
1460 IF DE=DIF THEN CALL HCHAR(12,A,32,1)
1470 IF DE=K THEN CALL HCHAR(12,A,32,1)
1480 NEXT A
1490 FOR A=1 TO 5 :: CALL SOUND(100,110+(110*A),0):: NEXT A :: CALL SOUND(400,44
0,0)
1500 GOTO 1630
1510 K=K+48
1520 FOR A=5 TO 29 STEP 3
1530 CALL GCHAR(12,A,DE)
1540 IF DE=K THEN CALL HCHAR(12,A,32,1)
```

```
1550 NEXT A
1560 FOR A=1 TO 5 :: CALL SOUND(100,110+(110*A),0):: NEXT A :: CALL SOUND(400,44
0,0)
1570 GOTO 1630
1580 BONUS=BONUS-1
1590 FOR A=1 TO 10 :: CALL SOUND(100,220-(A*10),0):: NEXT A :: CALL SOUND(400,11
0,0)
1600 IF BONUS=-1 THEN 1840
1610 DISPLAY AT(6,14)SIZE(1):USING "#":BONUS
1620 GOTO 980
1630 PUNKTE=PUNKTE+(BONUS*10*DURCHGANG)
1640 IF PUNKTE<=HS THEN 1670
1650 HS=PUNKTE
1660 GOSUB 1720
1670 IF PUNKTE>=10000 THEN 1690 ELSE IF PUNKTE>=1000 THEN 1700 ELSE IF PUNKTE>=1
00 THEN 1710
1680 DISPLAY AT(3,6)SIZE(2):USING "##":PUNKTE :: GOTO 980
1690 DISPLAY AT(3,3)SIZE(5):USING "#####":PUNKTE :: GOTO 980
1700 DISPLAY AT(3,4)SIZE(4):USING "####":PUNKTE :: GOTO 980
1710 DISPLAY AT(3,5)SIZE(3):USING "###":PUNKTE :: GOTO 980
1720 IF HS>=10000 THEN 1740 ELSE IF HS>=1000 THEN 1750 ELSE IF HS>=100 THEN 1760
1730 DISPLAY AT(3,25)SIZE(2):USING "##":HS :: RETURN
1740 DISPLAY AT(3,22)SIZE(5):USING "#####":HS :: RETURN
1750 DISPLAY AT(3,23)SIZE(4):USING "####":HS :: RETURN
1760 DISPLAY AT(3,24)SIZE(3):USING "###":HS :: RETURN
1770 W=48
1780 FOR A=5 TO 29 STEP 3
1790 W=W+1
1800 CALL HCHAR(12,A,W,1)
1810 NEXT A
1820 DURCHGANG=DURCHGANG+1 :: BONUS=4 :: DISPLAY AT(6,14)SIZE(1):USING "#":BONUS
1830 GOTO 1030
1840 DISPLAY AT(23,2)SIZE(22):"NOCH EIN SPIEL ? (J/N)"
1850 CALL KEY(0,K,S)
1860 IF K=74 THEN 1880 ELSE IF K<>78 THEN 1850
1870 CALL CLEAR :: CALL SCREEN(2):: FOR A=1 TO 2000 :: NEXT A :: END
1880 DISPLAY AT(23,2)SIZE(22):"                        "
1890 W1=48
1900 FOR A=5 TO 29 STEP 3
1910 W1=W1+1
1920 CALL HCHAR(12,A,W1,1)
1930 NEXT A
1940 PUNKTE=0 :: DISPLAY AT(3,3)SIZE(5):"     "
1950 GOTO 940
1960 CALL CHAR(97,"0044102844 7C4444",111,"004438444444438",117,"00440044444443
8")
1970 DISPLAY AT(3,5):"SINN DES SPIELES:"
1980 DISPLAY AT(8,1):"IN DIESEM SPIEL uBERNIMMT   DER COMPUTER DAS WERFEN      DE
R WuRFEL. MIT HILFE DER   GEWORFENEN SUMME MuSST       "
1990 DISPLAY AT(12,1):"IHR VERSUCHEN DIE ZAHLEN    VON 1 BIS 9 ZU LoSCHEN."
2000 GOSUB 2260
2010 DISPLAY AT(3,5):"LoSCHEN DER ZAHLEN:"
2020 DISPLAY AT(6,1):"ZUM LoSCHEN MuSSEN DIE      TASTEN 1-9 BENUTZT WERDEN.
                      BEISPIEL:"
2030 DISPLAY AT(10,1):"DER COMPUTER WuRFELT DIE    ZAHLEN 1 UND 6.              S
UMME IST GLEICH 7          FuR EUCH BESTEHEN DIE"
2040 DISPLAY AT(14,1):"MoGLICHKEITEN DIE ZAHL(EN)            1 UND 6
      2 UND 5                        3 UND 4"
2050 DISPLAY AT(18,1):"              7                ZU LoSCHEN." :: GOSUB 2260
2060 DISPLAY AT(6,1):"WIE IHR SEHT KANN MAN BEI   DIESER SUMME ENTWEDER ZWEI  ZA
HLEN ODER AUCH NUR EINE   ZAHL LoSCHEN."
```

TI-99

```
2070 DISPLAY AT(10,1):"WuRFELT DER COMPUTER EINE   SUMME VON 10 UND GRoSSER,   K
oNNEN LOGISCHERWEISE NUR   2 ZAHLEN GELoSCHT WERDEN."
2080 DISPLAY AT(14,1):"SOLLEN 2 ZAHLEN GELoSCHT    WERDEN, BRAUCHT NUR DIE     K
LEINERE ZAHL GEDRuCKT      WERDEN; DIE ANDERE LoSCHT"
2090 DISPLAY AT(18,1):"DER COMPUTER AUTOMATISCH." :: GOSUB 2260
2100 DISPLAY AT(3,5):"PUNKTEZaHLUNG:"
2110 DISPLAY AT(6,1):"DIE BONUSZAHL GIBT DIE      PUNKTE AN DIE IHR FuR DAS   Lo
SCHEN VON ZAHLEN BEKOMMT.                                  "
2120 DISPLAY AT(10,1):"PUNKTE=                     BONUSZAHL*DURCHGANG*10
                          FuR DAS LoSCHEN VON EINER"
2130 DISPLAY AT(14,1):"ZAHL GIBT ES DIE GLEICHEN   PUNKTE, WIE FuR DAS         L
oSCHEN VON ZWEI ZAHLEN." :: GOSUB 2260
2140 DISPLAY AT(3,5):"BONUSZAHL:"
2150 DISPLAY AT(6,1):"DIE BONUSZAHL KANN SICH IM  SPIEL aNDERN.                BE
I EINEM PASCH (2 GLEICHE  ZAHLEN 4/4 5/5 USW.) ER- "
2160 DISPLAY AT(10,1):"HoHT SICH DIE BONUSZAHL UM  ZWEI.                        B
ESTEHT KEINE MoGLICHKEIT   EINE ZAHL (ZWEI ZAHLEN)   "
2170 DISPLAY AT(14,1):"ZU LoSCHEN, KANN MIT DER    LEERTASTE NEU GEWuRFELT     W
ERDEN. DABEI ERNIEDRIGT    SICH DIE BONUSZAHL UM 1."
2180 DISPLAY AT(18,1):"BEI BONUSZAHL=0 HABT IHR    NUR NOCH EINE CHANCE;       S
OLLTE DIESE VERGEBEN WER-  DEN IST DAS SPIEL ZU ENDE."
2190 GOSUB 2260
2200 DISPLAY AT(3,1):"WEITERE ERLaUTERUNGEN:"
2210 DISPLAY AT(6,1):"NACHDEM EINE ZAHLENKOLON-   NE VOLLSTaNDIG GELoSCHT     WU
RDE, BEGINNT EIN NEUER    DURCHGANG. DIE BONUSZAHL   "
2220 DISPLAY AT(10,1):"BETRaGT WIEDER 4."
2230 DISPLAY AT(16,1):"UM DAS SPIELFELD AUFZU-     BAUEN BRAUCHT DER COMPUTER  Z
EIT.                       HABT ETWAS GEDULD!!! "
2240 GOSUB 2260
2250 CALL CHARSET :: CALL SCREEN(2):: GOTO 430
2260 DISPLAY AT(24,2):"ENTER DRuCKEN" :: CALL KEY(3,K,S):: IF K<>13 THEN 2260 EL
SE CALL CLEAR :: RETURN
```

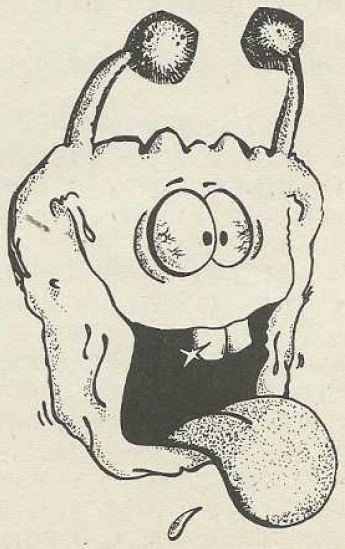
Dragon 32

Blizzard

Blizzards sind äußerst intelligente Wesen die Ihnen nach dem Leben trachten. Sie können sich jedoch verteidigen wenn Sie die im Raum verteilten Protonen benutzen. Mit Hilfe der Protonen graben Sie Ihre Fallen und versuchen Sie die Blizzards zu fangen. Mit dem rechten Joystick bewegen Sie die Spielfigur direkt unter ein Proton und drücken die Feuertaste. Nochmaliges Drücken läßt neben Ihnen ein Loch entstehen. Jetzt kommt es darauf an, den Blizzard in die Falle zu locken. Haben Sie das geschafft, decken Sie das Loch ganz schnell wieder ab, bevor sich der Blizzard befreien kann. Schafft er es trotzdem, ist Ihre Lebenserwartung gleich null.

Nach verlorenem Durchgang kommen Sie dennoch in die nächsthöhere Spielstufe. Allerdings haben Sie es jetzt mit zwei Blizzards zu tun, in der dritten Spielstufe mit drei Blizzards usw. Für Besitzer des Tardy Colo empfiehlt es sich die Poke-Werte und Adressen zu ändern.

Viel Spaß!



Karl der Käfer

Ein Spiel für den TI99 **ohne** Ext. Basic!

Karl der Käfer sucht sich eine neue Heimat. Auf dem Weg dorthin hat er jedoch viele Gefahren zu bestehen. Genaugenommen hat Karl drei Gefahren zu überwinden.

Er muß zuerst eine Gegend durchqueren, die ziemlich schnell mit Häusern vollgebaut wird. Mehr Häuser, weniger Fressen. Karl kann nur Überleben, wenn er noch genug Blätter findet und den Weg beschreitet, der ihn am schnellsten aus der Gefahrenzone bringt. – Die zweite Gefahr hat Karl zu bestehen, als er ein Teil des Weges im Flug bewältigen will. – Die dritte Gefahr bei Hindernissen, die sich ihm in den Weg stellen.

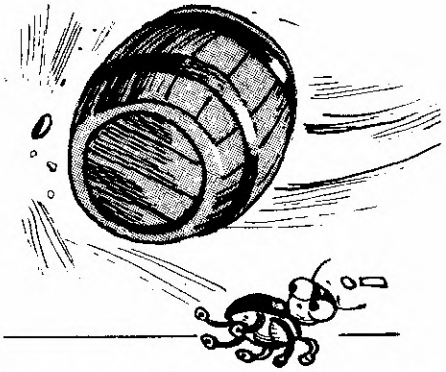
Aber verraten wir nicht zu viel. Das Spiel erklärt sich bei Beginn von selbst und kann über Tastatur gesteuert werden.

Wir können nur noch viel Spaß wünschen!

****

Programmaufbau:

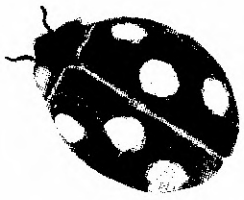
100– 190 Einleitung
200– 220 Schwierigkeitsgradwahl
230– 390 Bildwahl
400–1770 Bild 1
410– 490 Zeichendefinierung
500– 530 Farben
540– 620 Erklärung
630– 700 Bildaufbau
710– 790 Häusersetzen
800– 840 Blättersetzen
850–1420 Steuerung
1430–1480 Blatt gefressen
1490–1580 Häusersetzen
1590–1650 Käfer tot
1660–1680 Alle Käfer tot
1690–1770 Bild 1 geschafft
1780–2870 Bild 2
1780–1890 Zeichendefinierung
1900–1920 Farben
1930–2010 Erklärung
2020–2230 Bildaufbau
2240–2780 Steuerung
2790–2860 Bild 2 geschafft
2870–3830 Bild 3
2890–2920 Zeichendefinierung
2930–2970 Farben
2980–3040 Erklärung
3050–3180 Bildaufbau
3190–3500 Steuerung
3510–3600 Bewegung Hindernis
3610–3730 Hindernis geschafft
3740–3830 Bild 3 geschafft
3840–4130 Ende
3840–3850 Käfer gerettet
3860–3980 Punkte
3990–4130 Noch ein Spiel?



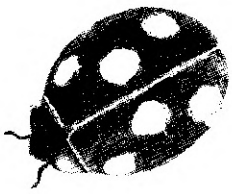
****

```
100 CALL CHAR(64,"3C4299A1A199423C")
110 CALL CLEAR
120 PRINT "****************************"
130 PRINT "**    KARL,DER KAEFER     ****                          **"
140 PRINT "**@ 1983 RALF MESCHEDE    **"
150 PRINT "****************************"::
160 PRINT "SIE SIND KARL,DER KAEFER UNDMUESSEN SICH EINE ANDERE    HEIMAT SUCHEN
.":
170 PRINT "SIE HABEN 3 LEBEN.":
180 PRINT "SIE MUESSEN AUF IHREM WEG   3 GEFAHREN UEBERWINDEN.":
190 PRINT "    VIEL ERFOLG!":
200 INPUT "  WAEHLEN SIE EIN SCHWIERIG-KEITSGRAD(1-3):":SKG
210 IF SKG>3 THEN 200
220 IF SKG<1 THEN 200
230 PRINT :::"MIT WELCHEN BILD WOLLEN SIE ANFANGEN ?":
```

```
240 GMB=0
250 PZ(1)=0
260 PZ(2)=0
270 PZ(3)=0
280 BO=0
290 P=0
300 TK=0
310 PRINT "BILD 1:....STADTDURCHQUERUNG"
320 PRINT "BILD 2:...........RISIKOFLUGBILD 3:........HINDERNISLAUF"
330 CALL KEY(0,T,Y)
340 IF Y=0 THEN 330
350 IF T=50 THEN 1780
360 IF T=51 THEN 2870
370 IF T=49 THEN 410
380 CALL SOUND(100,300,0)
390 GOTO 330
400 REM  BILD 1
410 NF=0
420 CALL CLEAR
430 BI=1
440 CALL CHAR(109,"2448F80404F84824")
450 CALL CHAR(108,"81825B24245B8281")
460 CALL CHAR(110,"24121F20201F1224")
470 CALL CHAR(111,"8141DA2424DA4181")
480 CALL CHAR(100,"10386CFE7C545C5C")
490 CALL CHAR(112,"0000183C3E1E0E01")
500 CALL SCREEN(4)
510 CALL COLOR(11,13,1)
520 CALL COLOR(9,2,1)
530 CALL COLOR(10,2,1)
540 PRINT "*****STADTDURCHQUERUNG*****"::
550 PRINT "SIE ALS no MUESSEN DURCH    EINE STADT GEHEN,DIE IMMER  MEHR MIT d ZU
GEBAUT WIRD."
560 PRINT ::"WENN SIE EIN Þ ESSEN KOENNENSIE 15 SCHRITTE WEITER GEHEN"::
570 PRINT "SIE SIND TOT,WENN SIE EIN d BERUEHREN ODER 16 SCHRITTE  GEHEN OHNE EI
N BLATT ZU     ESSEN."::
580 PRINT "SIE BEWEGEN SICH DURCH:"::
590 PRINT "          E-(OBEN)                S-(LINKS) D-(RECHTS)            X-(UN
TEN)"::
600 PRINT "DURCH DRUECKEN EINER TASTE  BEGINNT DAS SPIEL."
610 CALL KEY(0,T,Y)
620 IF Y=0 THEN 610
630 CALL CLEAR
640 NF=0
650 CALL HCHAR(1,3,100,28)
660 CALL HCHAR(24,3,100,28)
670 CALL VCHAR(3,3,100,21)
680 CALL VCHAR(2,30,100,19)
690 CALL HCHAR(2,2,110)
700 CALL HCHAR(2,3,111)
710 SK=3
720 ZK=2
730 IF SKG=1 THEN 800
740 FOR A=1 TO 12*SKG
750 RANDOMIZE
760 ZH=INT(RND*22+2)
770 SH=INT(RND*25+4)
780 CALL HCHAR(ZH,SH,100)
790 NEXT A
800 FOR A=4 TO 29
810 ZB=INT(RND*22+2)
820 CALL HCHAR(ZB,A,112)
```

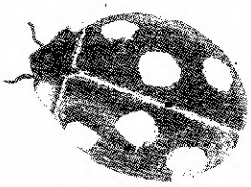


```
830 NEXT A
840 CALL SOUND(400,110,0)
850 CALL KEY(0,T,Y)
860 IF Y=0 THEN 1490
870 IF T=68 THEN 1070
880 IF T=69 THEN 1200
890 IF T=88 THEN 1330
900 IF T<>83 THEN 1490
910 CALL HCHAR(ZK,SK-1,32,2)
920 NF=NF+1
930 IF NF=15 THEN 1590
940 P=P-1
950 CK=109
960 SK=SK-1
970 IF SK>2 THEN 990
980 SK=3
990 CALL GCHAR(ZK,SK-1,FG)
1000 IF FG=32 THEN 1030
1010 IF FG=112 THEN 1430
1020 IF FG=100 THEN 1590
1030 CALL HCHAR(ZK,SK-1,108)
1040 CALL HCHAR(ZK,SK,109)
1050 CALL SOUND(100,400,0)
1060 GOTO 1490
1070 CALL HCHAR(ZK,SK-1,32,2)
1080 NF=NF+1
1090 IF NF=15 THEN 1590
1100 P=P-1
1110 CK=111
1120 SK=SK+1
1130 IF SK<31 THEN 1150
1140 GOTO 1690
1150 CALL GCHAR(ZK,SK,FG)
1160 IF FG=32 THEN 1190
1170 IF FG=112 THEN 1430
1180 IF FG=100 THEN 1590
1190 GOTO 1390
1200 CALL HCHAR(ZK,SK-1,32,2)
1210 NF=NF+1
1220 IF NF=15 THEN 1590
1230 P=P-1
1240 ZK=ZK-1
1250 CALL GCHAR(ZK,SK,FG)
1260 CALL GCHAR(ZK,SK-1,GF)
1270 IF FG=100 THEN 1590
1280 IF GF=100 THEN 1590
1290 IF GF=112 THEN 1430
1300 IF FG=112 THEN 1430
1310 IF FG=32 THEN 1390
1320 IF GF=32 THEN 1390
1330 CALL HCHAR(ZK,SK-1,32,2)
1340 NF=NF+1
1350 IF NF=15 THEN 1590
1360 P=P-1
1370 ZK=ZK+1
1380 GOTO 1250
1390 CALL HCHAR(ZK,SK-1,CK-1)
1400 CALL HCHAR(ZK,SK,CK)
1410 CALL SOUND(100,400,0)
1420 GOTO 1490
1430 NF=0
1440 P=P+60
```

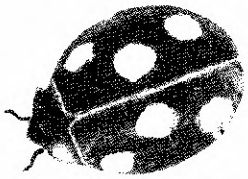


```
1450 CALL SOUND(200,220,0)
1460 CALL SOUND(100,550,0)
1470 CALL SOUND(200,480,0)
1480 GOTO 1390
1490 RANDOMIZE
1500 ZH=INT(RND*22+2)
1510 SH=INT(RND*25+4)
1520 CALL GCHAR(ZH,SH,FG)
1530 IF FG=108 THEN 1500
1540 IF FG=109 THEN 1500
1550 IF FG=110 THEN 1500
1560 IF FG=111 THEN 1500
1570 CALL HCHAR(ZH,SH,100)
1580 GOTO 850
1590 PRINT ::::::::::::::::::"EIN KAEFER IST TOT!"
1600 CALL SOUND(400,-6,0)
1610 FOR A=1 TO 200
1620 NEXT A
1630 TK=TK+1
1640 IF TK=3 THEN 1660
1650 ON BI GOTO 630,2020,3050
1660 PRINT ::::::::::::::::::"ALLE DREI KAEFER SIND TOT!"::
1670 PZ(BI)=P
1680 GOTO 3870
1690 CALL SOUND(100,300,0)
1700 CALL SOUND(200,500,0)
1710 P=P+500
1720 CALL SOUND(200,200,0)
1730 PZ(1)=P
1740 P=0
1750 CALL SOUND(200,400,0)
1760 GMB=GMB+1
1770 IF GMB=3 THEN 3840
1780 REM  BILD 2
1790 BI=2
1791 CALL CLEAR
1800 IF SKG=3 THEN 1830
1810 KGG=1
1820 GOTO 1840
1830 KGG=2
1840 CALL CHAR(104,"1F20183E3E18201F")
1850 CALL CHAR(105,"01815A3C3C5A8101")
1860 CALL CHAR(110,"24121F20201F1224")
1870 CALL CHAR(111,"8141DA2424DA4181")
1880 CALL CHAR(103,"AA552A542A1408")
1890 CALL CHAR(119,"FFFFFFFFFFFFFFFFFFFF")
1900 CALL COLOR(11,6,6)
1910 CALL COLOR(10,2,1)
1920 CALL COLOR(9,2,6)
1930 PRINT "*********RISIKOFLUG*********"::
1940 PRINT "SIE MUESSEN DURCH 2 LUFT-   STROEME FLIEGEN,DIE DURCH 9 VERSEUCHT SI
ND."::
1950 PRINT "SIE WERDEN,WENN SIE IN EINEMLUFTSTROM SIND,NACH OBEN    ABGETRIEBEN.
"::
1960 PRINT "SIE KOENNEN ZWISCHEN DEN    BEIDEN LUFTSTROEMEN NEUE    KRAEFTE SAMM
ELN."
1970 PRINT "SIE STERBEN,WENN SIE IN EINEGIFTWOLKE FLIEGEN ODER DEN  OBEREN(UNTER
EN) BILDSCHIRM- RAND BERUEHREN."
1980 PRINT "SIE FLIEGEN DURCH:                                      E-NACH OBEN
                X-NACH UNTEN"
1990 PRINT "DURCH DRUECKEN EINER TASTE  BEGINNT DAS SPIEL."
2000 CALL KEY(0,T,Y)
```

```
2010 IF Y=0 THEN 2000
2020 CALL CLEAR
2030 P=P+100
2040 FOR A=1 TO 24
2050 GOSUB 2080
2060 NEXT A
2070 GOTO 2170
2080 PRINT "  ωωωωωωωωωωωω     ωωωωωωωωωωωωωω"
2090 RANDOMIZE
2100 SL1=INT(RND*10+5)
2110 CALL HCHAR(23,SL1,103)
2120 FOR AA=1 TO KGG
2130 SL2=INT(RND*11+19)
2140 CALL HCHAR(23,SL2,103)
2150 NEXT AA
2160 RETURN
2170 SK=3
2180 ZK=15
2190 CALL COLOR(10,2,1)
2200 CALL HCHAR(15,2,110)
2210 CALL HCHAR(15,3,111)
2220 KK=111
2230 CALL SOUND(400,110,0)
2240 CALL KEY(0,T,Y)
2250 IF Y=0 THEN 2670
2260 IF T=69 THEN 2570
2270 IF T<>88 THEN 2660
2280 IF SK>5 THEN 2320
2290 LC=32
2300 CALL COLOR(10,2,1)
2310 GOTO 2370
2320 IF SK<16 THEN 2350
2330 IF SK>18 THEN 2350
2340 GOTO 2290
2350 LC=119
2360 CALL COLOR(10,2,6)
2370 IF T=69 THEN 2580
2380 CALL HCHAR(ZK,SK-1,LC,2)
2390 KK=105
2400 ZK=ZK+1
2410 P=P-1
2420 CALL SOUND(100,500,0)
2430 IF ZK>23 THEN 2780
2440 SK=SK+1
2450 IF SK=4 THEN 2480
2460 IF SK<16 THEN 2510
2470 IF SK>18 THEN 2510
2480 KK=111
2490 IF SK=31 THEN 2790
2500 GOTO 2560
2510 IF SK=31 THEN 2790
2520 CALL GCHAR(ZK+1,SK-1,FG)
2530 CALL GCHAR(ZK+1,SK,GF)
2540 IF FG=103 THEN 2780
2550 IF GF=103 THEN 2780
2560 GOTO 2670
2570 GOTO 2280
2580 KK=105
2590 CALL HCHAR(ZK,SK-1,LC,2)
2600 ZK=ZK-2
2610 CALL SOUND(100,500,0)
2620 P=P-1
```



```
2630 IF ZK<1 THEN 2780
2640 SK=SK+1
2650 GOTO 2450
2660 Y=0
2670 IF KK=105 THEN 2690
2680 CALL HCHAR(ZK,SK-1,32,2)
2690 GOSUB 2080
2700 IF KK=111 THEN 2720
2710 IF Y=0 THEN 2750
2720 CALL HCHAR(ZK,SK-1,KK-1)
2730 CALL HCHAR(ZK,SK,KK)
2740 GOTO 2240
2750 ZK=ZK-1
2760 IF ZK<1 THEN 2780
2770 GOTO 2240
2780 GOTO 1590
2790 CALL SOUND(100,300,0)
2800 CALL SOUND(100,500,0)
2810 CALL SOUND(100,400,0)
2820 P=P+500
2830 PZ(2)=P
2840 P=0
2850 GMB=GMB+1
2860 IF GMB=3 THEN 3840
2870 BI=3
2880 REM  3.BILD
2890 CALL CHAR(115,"FFFF")
2900 CALL CHAR(99,"0000006060606060")
2910 CALL CHAR(106,"00000718204 03F22")
2920 CALL CHAR(107,"0102856E1212EC20")
2930 CALL CLEAR
2940 CALL COLOR(9,2,1)
2950 CALL COLOR(11,5,1)
2960 CALL SCREEN(4)
2970 CALL COLOR(10,2,1)
2980 PRINT "*******HINDERNISLAUF********":::
2990 PRINT "SIE jk MUESSEN UEBER c       SPRINGEN,DAS IHNEN ENTGEGEN KOMMT.":::
3000 PRINT "WENN SIE EIN HINDERNIS       NICHT UEBERSPRINGEN,SIND SIETOT.":::
3010 PRINT "WENN SIE DIE E-TASTE         DRUECKEN,SPRINGEN SIE.":::
3020 PRINT "DURCH DRUECKEN EINER TASTE   BEGINNT DAS SPIEL.":::::
3030 CALL KEY(0,T,Y)
3040 IF Y=0 THEN 3030
3050 CALL CLEAR
3060 FOR A=8 TO 24 STEP 8
3070 CALL HCHAR(A,1,115,32)
3080 CALL HCHAR(A-1,32,99)
3090 NEXT A
3100 SK=2
3110 ZK=7
3120 HS=32
3130 HZ=7
3140 P=P+150
3150 CALL HCHAR(ZK,SK-1,106)
3160 CALL HCHAR(ZK,SK,107)
3170 R=1
3180 CALL SOUND(400,110,0)
3190 CALL KEY(0,T,Y)
3200 IF Y=0 THEN 3420
3210 IF T<>69 THEN 3420
3220 P=P-4
3230 CALL HCHAR(ZK,SK-1,32,2)
3240 CALL SOUND(100,500,0)
```



```
3250 ZK=ZK-1
3260 SK=SK+2
3270 IF SK>31 THEN 3610
3280 CALL HCHAR(ZK,SK-1,106)
3290 CALL HCHAR(ZK,SK,107)
3300 CALL SOUND(200,300,0)
3310 CALL HCHAR(ZK,SK-1,32,2)
3320 ZK=ZK+1
3330 SK=SK+2
3340 IF SK>31 THEN 3610
3350 CALL GCHAR(ZK,SK-1,FG)
3360 CALL GCHAR(ZK,SK,GF)
3370 IF FG<>32 THEN 4130
3380 IF GF<>32 THEN 4130
3390 CALL HCHAR(ZK,SK-1,106)
3400 CALL HCHAR(ZK,SK,107)
3410 GOTO 3510
3420 CALL HCHAR(ZK,SK-1,32,2)
3430 P=P-1
3440 CALL SOUND(200,400,0)
3450 SK=SK+1
3460 IF SK>31 THEN 3610
3470 CALL GCHAR(ZK,SK,FG)
3480 IF FG<>32 THEN 4130
3490 CALL HCHAR(ZK,SK-1,106)
3500 CALL HCHAR(ZK,SK,107)
3510 FOR ZX=1 TO R
3520 CALL HCHAR(HZ,HS,32)
3530 HS=HS-1
3540 IF HS>1 THEN 3560
3550 HS=31
3560 CALL GCHAR(HZ,HS,GF)
3570 IF GF<>32 THEN 4130
3580 CALL HCHAR(HZ,HS,99)
3590 NEXT ZX
3600 GOTO 3190
3610 CALL SOUND(200,200,0)
3620 CALL SOUND(200,500,0)
3630 CALL SOUND(200,350,0)
3640 IF ZK>10 THEN 3670
3650 ZK=15
3660 GOTO 3690
3670 IF ZK>20 THEN 3740
3680 ZK=23
3690 R=SKG
3700 SK=2
3710 HZ=HZ+8
3720 HS=32
3730 GOTO 3190
3740 FOR XA=1 TO 3
3750 FOR XF=110 TO 660 STEP 110
3760 CALL SOUND(100,XF,0)
3770 NEXT XF
3780 NEXT XA
3790 P=P+600
3800 PZ(3)=P
3810 GMB=GMB+1
3820 P=0
3830 IF GMB<3 THEN 410
3840 PRINT "    ICH GRATULIERE!"::
3850 BO=400*(3-TK)
3860 PRINT "SIE HABEN ALLE DREI BILDER  BEI SCHWIERIGKEITSGRAD";SKG,"GESCHAFFT."
::
```

```
3870 PRINT "BILD 1:";PZ(1);"PUNKTE"
3880 PRINT "BILD 2:";PZ(2);"PUNKTE"
3890 PRINT "BILD 3:";PZ(3);"PUNKTE"::
3900 PRINT "BONUS :";BO;"PUNKTE"::
3910 P=PZ(1)+PZ(2)+PZ(3)+BO
3920 PRINT "PUNKTE INSGESAMT:";P
3930 IF P<TB THEN 3980
3940 CALL SOUND(100,300,0)
3950 CALL SOUND(200,600,0)
3960 CALL SOUND(100,400,0)
3970 TB=P
3980 PRINT "TAGESBESTLEISTUNG:";TB
3990 PRINT :::"NOCH EIN SPIEL? (J/N)"
4000 CALL KEY(0,T,Y)
4010 IF Y=0 THEN 4000
4020 IF T=74 THEN 110
4030 IF T=78 THEN 4060
4040 CALL SOUND(200,110,0)
4050 GOTO 4000
4060 PRINT ::::"    BIS BALD!                    © 1983 RALF MESCHEDE"
4070 FOR T=1 TO 7
4080 FOR F=110 TO 990 STEP 110
4090 CALL SOUND(200,F,T)
4100 NEXT F
4110 NEXT T
4120 END
4130 GOTO 1590
```

LASER®

LASER 110 – 4 KByte RAM. Dieser „persönliche Computer" eignet sich als Helfer beim Studium, am Arbeitsplatz oder bei Analysen und Statistiken. Mit seinen Peripheriegeräten erfüllt er alle Ansprüche an einen Home-Computer.

ALIEN-LANDING

Das Programm ist in Extended-Basic geschrieben und benötigt Joysticks.

Angriff aus dem All.

Versuchen Sie mit einer beweglichen Kanone »ALIENS« abzuschießen, die auf der Planetenoberfläche landen wollen.
Sie haben pro Runde 10 »ALIENS« zu treffen, aber nur 25 Raketen und 180 sec Zeit.

Ihre Aufgabe wird dadurch erschwert, daß die »ALIENS« pro Runde immer intelligenter und schneller werden. Bei steigender Schwierigkeit haben Sie wirklich keine Sekunde Zeit zu verlieren, denn Sie müssen immer damit rechnen, daß Ihre Kanone von einem Monteur repariert werden muß, der zu den unmöglichsten Zeiten in Aktion tritt, und Ihre Restzeit dadurch leider verringert.

Auch hat die Anzahl der verschossenen Raketen eine Einwirkung auf die Punktvergabe pro Treffer.

Noch ein Hinweis zum Abtippen des Programms:

Einige Befehlszeilen werden beim normalen Abtippen nicht vollständig angenommen. Beenden Sie die Eingabe mit ⟨ENTER⟩ und rufen Sie diese Programmzeile mit ⟨FNCT-REDO⟩ zurück, wonach Sie dann die Zeile komplett fertig schreiben können. ⟨ENTER⟩ nicht vergessen.

Ebenfalls müssen alle Ausrufezeichen mit eingegeben werden.

Aus Platzgründen wurde auf erklärende Remarks verzichtet.

Wenn das Programm läuft, wird der gesamte Speicherbereich bis auf ca. 100 Bytes ausgenützt.

Wir finden, ein tolles Spiel, und wünschen viel Spaß und Punkte.

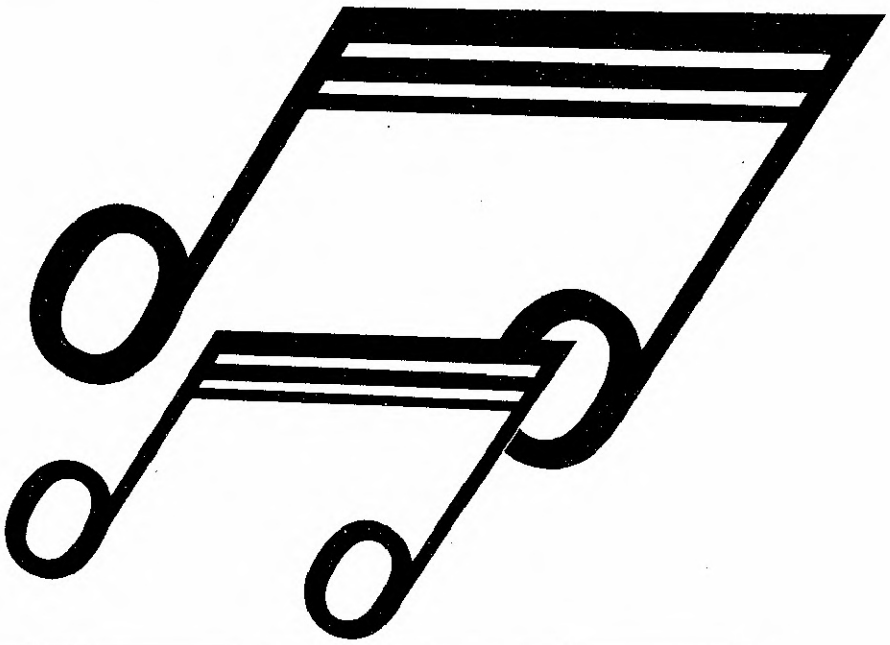
****

```
100 CALL CLEAR :: CALL SCREEN(2):: FOR I=1 TO 12 :: CALL COLOR(I,16,2):: NEXT I
:: RANDOMIZE
110 CALL MAGNIFY(2):: I=1 :: FOR T=1 TO 13 :: READ A :: CALL SPRITE(#T,A,INT(RND
*13)+3,I*16+2,I*3+80,0,-9):: I=I+1 :: IF A=32 THEN I=1
120 NEXT T
130 DISPLAY AT(9,1)BEEP:"***** ALIEN - LANDING  *****": : : :"** written by W.
Doeltsch  **": :" **  6108 Weiterstadt 3  **": :"  **  Rappmuehlstr.58c  **"
140 GOSUB 1400 :: DISPLAY AT(5,8)BEEP:"Verhindern Sie die Landung": :"      von
10 Alien's.": : :"Nach 10 Abschuessen wird es": :"schwerer."
150 DISPLAY AT(15,1):"Es arbeitet Joystick Nr.1": :"fuer links/rechts und Feuer.
" :: GOSUB 1400
160 CALL CHAR(128,"010",136,"000000001",92,"18",34,"00008",33,"FFFFFFFFFFFFFFFF"
)
170 CALL CHAR(35,"0102041820404080",36,"80C060783C3E0E01",37,"81C1D37B7E1E0C00")
180 DATA 65,76,73,69,78,32,76,65,78,68,73,78,71
190 DATA 20,2,37,22,2,35,20,3,37,21,3,35,20,4,36,21,5,36,22,6,36,22,7,35,21,8,35
,21,9,37
200 DATA 20,10,35,20,11,37,21,11,36,20,12,36,22,12,36,21,13,37,21,14,36,21,15,35
,21,16,37
210 DATA 21,17,36,22,18,36,22,19,35,21,20,35,20,21,35,20,22,37,21,22,36,20,23,37
,22,23,36
220 DATA 20,24,37,20,25,36,21,26,36,22,27,36,21,28,35,20,29,35,20,30,36,21,31,37
,21,15,35,21,26,36
230 KA$="0101010101010323262E3F0163B5DF6080808080808080C0C46474FC80C6ABFD06"
240 E1$="0209200421880150001280200440080020084400210800248100240094011080"
250 F2$="0000000012000208020005200200000000000090005004400090401400400000"
260 E3$="0000000500020800 0A0005000000000000000004000A000104000200000000" :: CA
LL CHAR(140,RPT$("0",16)&"1818081838180818"&RPT$("0",32))
270 CALL CHAR(44,RPT$("0",16)&"1818887E1A192422"&RPT$("0",32)):: CALL CHAR(132,R
PT$("0",16)&"1818085C3A181414"&RPT$("0",32))
280 CALL CHAR(108,KA$):: CALL CHAR(116,"0000000000000FFFF"):: CALL CHAR(112,E1$):
: CALL CHAR(120,E2$):: CALL CHAR(124,E3$)
290 R$="0101010103070504000500020102000180808080C0E0A0208020408040008040" :: CAL
L CHAR(40,R$)
300 CALL COLOR(13,14,1,1,6,1,14,5,1,8,12,1,11,14,1):: CALL CLEAR :: CALL HCHAR(2
3,1,116,64):: GOSUB 1280
310 FOR I=1 TO 38 :: READ X,Y,Z :: CALL HCHAR(X,Y,Z):: NEXT I
320 CALL VCHAR(1,1,33,24):: CALL VCHAR(1,32,33,24):: PU=0 :: LE=1 :: GOSUB 1440
330 CALL HCHAR(1,1,33,32):: CALL HCHAR(24,1,33,32):: CALL DELSPRITE(ALL)
340 CALL MAGNIFY(3):: L1=166 :: L2=126 :: TREF,FEHL,T=0 :: U4=16 :: ON LE GOTO 9
80,1000,1020,1040,1060,1080,1100,1120,1140,1160
350 CALL CHAR(96,U1$,100,U2$,104,U3$)
360 DISPLAY AT(1,1)SIZE(13):"!!!UFO NR: 0 " :: DISPLAY AT(24,1)SIZE(13):"RAKETE
NR: 0"
370 DISPLAY AT(24,17)SIZE(11):"PUNKTE:     " :: DISPLAY AT(24,23)SIZE(6):PU :: DI
SPLAY AT(1,17)SIZE(13):"TIME:     SEC" :: CALL DELSPRITE(#1,#2,#3,#4,#5,#6):: CAL
L MAGNIFY(3)
380 FOR I=30 TO 0 STEP -2 :: CALL SOUND(100,600,I,700,I,-2,I):: FOR D=1 TO 25 ::
 NEXT D :: NEXT I
390 GOSUB 1440 :: CALL SPRITE(#1,96,F,INT(RND*(95-45+1))+45,256,-20,INT(RND*30)+
95*SGN(RND-.5))
400 CALL SPRITE(#2,108,11,L1,L2):: CALL COLOR(#1,F):: CALL MOTION(#1,INT(RND*3)+
1,INT(RND*20)+10*SGN(RND-.5))
410 FOR I=96 TO 104 STEP 4 :: CALL PATTERN(#1,I):: CALL JOYST(1,A,B):: CALL MOTI
ON(#2,0,6*A):: CALL KEY(1,X,Y):: IF X=18 THEN 470 :: GOSUB 1460
420 CALL JOYST(1,A,B):: CALL MOTION(#2,0,6*A):: CALL KEY(1,X,Y):: IF X=18 THEN 4
70
430 NEXT I
440 T=T+1.12 :: GOSUB 1450 :: IF T>180 THEN 890 :: CALL PATTERN(#1,96):: CALL JO
YST(1,A,B):: CALL MOTION(#2,0,6*A):: CALL KEY(1,X,Y):: IF X=18 THEN 470
450 CALL MOTION(#1,RND*LE,INT(RND*LEV+(LEV/2))*SGN(RND-.4)):: CALL PATTERN(#1,10
```

```
4):: CALL JOYST(1,A,B):: CALL MOTION(#2,0,6*A)
460 CALL KEY(1,X,Y):: IF X=18 THEN 470 :: GOTO 410
470 CALL MOTION(#2,0,0):: CALL POSITION(#2,P1,P2):: CALL SPRITE(#3,40,16,P1+4,P2
,-128,0):: CALL SOUND(-50,-5,0):: GOTO 920
480 CALL SOUND(-1000,-7,0):: CALL PATTERN(#3,112):: CALL MOTION(#3,-4,0):: CALL
POSITION(#1,P1,P2)
490 CALL PATTERN(#1,112):: CALL COLOR(#1,F):: CALL MOTION(#1,-5,2):: CALL SOUND(
-1000,-7,2)
500 FOR I=4 TO 7 :: CALL SPRITE(#I,112,F,P1+10,P2,I*-10,0):: NEXT I :: CALL MOTI
ON(#3,3,0,#1,2,0):: CALL SOUND(-1000,-7,3)
510 CALL MOTION(#4,5,-1,#5,5,1,#6,3,2,#7,3,-2):: CALL SOUND(-1000,-7,6)
520 FOR I=8 TO 11 :: CALL SPRITE(#I,112,F,P1+10,P2,I*-8,0):: NEXT I :: CALL SOUN
D(-1000,-7,8)
530 CALL MOTION(#8,6,-2,#9,6,2,#10,2,2,#11,2,-2,#3,4,0):: CALL SOUND(-1000,-7,10
)
540 FOR I=12 TO 15 :: CALL SPRITE(#I,112,F,P1+10,P2,I*-5,0):: NEXT I :: CALL SOU
ND(-1000,-7,12)
550 CALL MOTION(#12,6,1,#13,3,-1,#14,3,-1,#15,3,1,#3,4,0):: CALL SOUND(-1000,-7,
14)
560 FOR I=16 TO 19 :: CALL SPRITE(#I,112,F,P1+10,P2,I*-3,0):: NEXT I :: CALL SOU
ND(-1000,-7,16)
570 CALL MOTION(#16,4,0,#17,3,-1,#18,3,1,#19,4,1):: CALL SOUND(-1000,-7,18):: T=
T+2.20 :: GOSUB 1450 :: GOSUB 1300 :: T=T+2.25 :: GOSUB 1450
580 TREF=TREF+1 :: FEHL=FEHL+1 :: DISPLAY AT(1,11)SIZE(3):TREF :: DISPLAY AT(24,
11)SIZE(3):FEHL
590 PU=PU+1010-(FEHL*10):: DISPLAY AT(24,23)SIZE(6):PU
600 CALL SOUND(-1000,-7,16):: FOR I=3 TO 11 :: CALL PATTERN(#I,120):: NEXT I ::
GOSUB 1300
610 CALL SOUND(-1000,-7,18):: FOR I=12 TO 19 :: CALL PATTERN(#I,120):: NEXT I ::
 GOSUB 1300
620 CALL SOUND(-1000,-7,20):: FOR I=3 TO 11 :: CALL PATTERN(#I,124):: NEXT I ::
GOSUB 1300
630 CALL SOUND(-1000,-7,22):: FOR I=12 TO 19 :: CALL PATTERN(#I,124):: NEXT I ::
 GOSUB 1300
640 CALL SOUND(-1000,-7,24):: CALL DELSPRITE(#1,#3,#4,#5,#6,#7,#8,#9,#10,#11,#12
,#13,#14,#15,#16,#17,#18,#19):: GOSUB 1300
650 T=T+4.35 :: GOSUB 1450 :: IF TREF=10 THEN 700 ELSE IF FEHL=25 THEN 860 ELSE
IF T>180 THEN 890 :: CALL SPRITE(#2,108,11,L1,L2)
660 RANDOMIZE :: IF INT(RND*8)<>1 THEN 670 ELSE GOSUB 1320 :: T=T+14.78 :: GOSUB
 1450 :: IF T>180 THEN 890
670 FOR X=30 TO 1 STEP -1.5 :: CALL SOUND(-99,-5,X):: NEXT X :: T=T+2.50 :: GOSU
B 1450 :: CALL COLOR(1,F,1)
680 GOSUB 1440 :: CALL PATTERN(#1,96):: CALL LOCATE(#1,INT(RND*(95-45+1))+45,256
)
690 CALL MOTION(#1,-20,INT(RND*30)+95*SGN(RND-.5)):: F=INT(RND*11+3):: CALL MOTI
ON(#1,INT(RND*-6),INT(RND*20)+10*SGN(RND-.5)):: T=T+.8 :: GOSUB 1450 :: GOTO 450
700 CALL DELSPRITE(ALL):: FOR I=1 TO 10 :: CALL SPRITE(#I,96,INT(RND*13)+3,96,12
8,INT(RND*110)*SGN(RND-.5),INT(RND*110)*SGN(RND-.5))
710 FOR X=1400 TO 1460 STEP 32 :: CALL SOUND(-99,X,9,X+4,9,X+8,9):: NEXT X :: GO
SUB 1300 :: NEXT I
720 FOR X=1 TO 2 :: CALL SOUND(200,440,1):: CALL SOUND(200,554,1):: CALL SOUND(2
00,440,1):: CALL SOUND(200,660,1)
730 CALL SOUND(700,880,0,660,5,554,10):: NEXT X
740 DISPLAY AT(1,1)SIZE(28):"!!!!!!!!!!GESCHAFFT!!!!!!!!!"
750 FOR I=1 TO 15 :: GOSUB 1300 :: CALL COLOR(1,INT(RND*13+3),1):: NEXT I :: GOT
O 770
760 DISPLAY AT(1,1)SIZE(28)BEEP:"!!!!!!!!!!VERLOREN!!!!!!!!!!" :: FOR I=1 TO 10
:: GOSUB 1300 :: NEXT I :: GOTO 800
770 LE=LE+1 :: IF LE=11 THEN 800 :: DISPLAY AT(24,1)SIZE(32)BEEP:"!!!!ES BEGINNT
 RUNDE !!!!!!!!!!!" :: DISPLAY AT(24,21)SIZE(3):LE
780 FOR I=1 TO 10 :: CALL MOTION(#I,0,(I*-9)+-6):: GOSUB 1300 :: CALL SOUND(-340
```

```
0,110+(I*3),I*2.5,113+(I*3),I*2.5,116+(I*3),I*2,-2,I*2.5):: NEXT I
790 FOR I=1 TO 4 :: GOSUB 1300 :: NEXT I :: GOTO 330
800 DISPLAY AT(24,1)SIZE(29)BEEP:"!!!!NEUES SPIEL? <J><N>!!!!!" :: GOSUB 1310
810 IF PU<=NPU THEN 830
820 NPU=PU
830 DISPLAY AT(8,17)SIZE(6):PU :: DISPLAY AT(12,17)SIZE(6):NPU
840 CALL KEY(2,T,S):: GOSUB 1300 :: IF S=0 THEN 840 :: IF T<>2 THEN 1270
850 DISPLAY AT(8,3):RPT$((CHR$(136)&CHR$(92)&CHR$(34)),8):: DISPLAY AT(12,3):RPT
$((CHR$(34)&CHR$(136)&CHR$(92)),8):: GOTO 320
860 CALL DELSPRITE(ALL):: DISPLAY AT(24,1)BEEP:"!!LEIDER KEINE RAKETEN MEHR!" ::
 FOR I=1 TO 10 :: GOSUB 1300 :: NEXT I
870 DISPLAY AT(24,1)BEEP:"IHRE NERVEN WAREN NICHT GUT!" :: FOR I=1 TO 10 :: GOSU
B 1300 :: NEXT I :: DISPLAY AT(24,1)BEEP:"!!IHR AUGENMASS AUCH NICHT!!"
880 CALL MAGNIFY(2):: GOSUB 1410 :: GOTO 760
890 CALL DELSPRITE(ALL):: DISPLAY AT(24,1)BEEP:"!TIME OUT!TIME OUT!TIME OUT!" ::
 FOR I=1 TO 10 :: GOSUB 1300 :: NEXT I
900 DISPLAY AT(24,1)BEEP:"!!!ACHTEN SIE AUF DIE UHR!!!" :: FOR I=1 TO 10 :: GOSU
B 1300 :: NEXT I :: DISPLAY AT(24,1)BEEP:"!!UND TREFFEN SIE SCHNELLER!"
910 CALL MAGNIFY(2):: GOSUB 1410 :: GOTO 760
920 FOR I=1 TO 9 :: CALL COINC(#1,#3,11,H):: IF H THEN 480
930 NEXT I
940 CALL DELSPRITE(#3):: GOSUB 1460
950 FEHL=FEHL+1 :: DISPLAY AT(24,11)SIZE(3):FEHL :: T=T+1.1 :: GOSUB 1460 :: GOS
UB 1450
960 PU=PU-10-(FEHL*5):: IF PU<0 THEN PU=0 :: DISPLAY AT(24,23)SIZE(5):PU :: IF T
>180 THEN 890 ELSE IF FEHL=25 THEN 860 :: GOSUB 1460
970 DISPLAY AT(24,23)SIZE(5):PU :: IF T>180 THEN 890 ELSE IF FEHL=25 THEN 860 ::
 GOTO 410
980 U1$="0103070F19393F77FBFC7F7F2F2750888C0E0F0989CFCEEDF3FFEFEF4E40A11" :: U2
$="000001071F39797FF0FF7F7F2F2750880000080E0F09C9EFE0FFFFEFEF4E40A11"
990 U3$="000000000071F79F9FF7E7D2B2750880000000000E0F89E9FFF7EBED4E40A11" :: LE
V=7 :: GOTO 350
1000 U1$="1117054D5F1FBFBF1F1010202050500088E8A0B2FAF8FDFDF8080804040A0A00" :: U
2$="1107474F1999BF3F1F2444480814140088E0E2F29899FDFCF82422121028280 0"
1010 U3$="0147470F9F9F31311F111122220505008 0E2E2F0F9F98C8CF888884444A0A000" :: L
EV=8 :: GOTO 350
1020 U1$="1028448383C7EFFFFFFFFF7CBB5428100814 22C1C1E3F7FFFFFFFF3EDD2A1408" :: U
2$="00081422414 1E3F7FFFFFF7CBB5428100010284482 82C7EFFFFFFF3EDD2A1408"
1030 U3$="0000040A11212171FBFFFF7CBB5428100000205088848487DFFFFF3EDD2A1408" :: L
EV=9 :: GOTO 350
1040 U1$="00000001F3070F0F0F07FB0202020 40E00000080DFE0F01010E0DF4040402070" :: U
2$="000080611B070F0C0C071B628204040E00000186D8E0F03030E0D8464120 2070"
1050 U3$="0040201108070F0808070B122644881C00020488D0E0F0F0F0E0D0486422113 8" :: L
EV=10 :: GOTO 350
1060 U1$="071C1933674FFF8F8F7F2E9B47300E00E0F8F8FCFEFEFF1F1FFEECB8E2084000" :: U
2$="071E1E3E7E7EFFE3E37F371D07100F00E078787C7E7EFFC7C7FE74D8E008F000"
1070 U3$="071F1F3F7F7FFFF8F87F3B0E47100200E03898CCE6F2FFF1F1FEB879E20C7000" :: L
EV=11 :: GOTO 350
1080 U1$="03070F19397F7C3F1F0C1E000000000C0E0F0989CFE3EFCF830780000000000" :: U
2$="03070F1F397F7E3E1F0C0C0C1E000000C0E0F0F89CFE7E7CF830303078000000"
1090 U3$="03070F1F3D7F7F3E1F0C0C0C0C0C0C1EC0E0F0F8BCFEFE7CF8303030303 03078" :: L
EV=12 :: GOTO 350
1100 U1$="01061CFF9C9CFF1F4701240922044012806038FFE7E7FFF8E284104208A10882" :: U
2$="01061CFFCECEFF1F0721041144012400806038FF7373FFF8E0842A005402 5000"
1110 U3$="01061CFFE7E7FF1F4709210A411400008806038FF3939FFF8E2882450045000 00" :: L
EV=13 :: GOTO 350
1120 U1$="80C0E4743C3C1E1F0F0D0503010101000103272E3C3C78F8F0B0A0C080808000" :: U
2$="0464F4BE3F0F0D0503010101000000000202 62F7DF9F0B0A0C080808000000000"
1130 U3$="183D7FF5E5E3C1C1E1703818080000001 8BCFEAFA7C7838387 0E1C1810000000" :: L
EV=14 :: GOTO 350
1140 U1$="0000000000000107060701000000000000000000000080E060E080000000000" :: U
```

```
2$="0000000001020F0E0E0F0301000000000000000080C0F07070F0C0800000000"
1150 U3$="0103070F1FFFFCFCFCFCFF1F0F07030180C0E0F0F8FF3F3F3F3FFFF8F0E0C080" :: L
EV=15 :: GOTO 350
1160 U1$="F1F1F1F1FF96B294B696FF7171710000C7C7C7C7FF99ABA9AB99FFC7C7C70000" :: U
2$="1C1C1C1CFF96B294B696FF1C1C1C0000F1F1F1F1FF99ABA9AB99FF7171710000"
1170 U3$="38383838FF96B294B696FF3838380000E3E3E3E3FF99ABA9AB99FFE3E3E30000" :: L
EV=16 :: GOTO 350
1180 CALL MOTION(#1,0,0,#2,0,0):: CALL SOUND(-2000,-5,10):: FOR I=1 TO 26 :: CAL
L POSITION(#1,U3,U4):: IF U3>166 THEN U3=166 :: GOTO 1200 ELSE U3=U3+1 :: CALL L
OCATE(#1,U3,U4)
1190 NEXT I
1200 CALL LOCATE(#1,U3,U4):: CALL PATTERN(#1,104):: CALL COINC(#1,#2,16,HIT):: I
F HIT THEN 1240 :: CALL SOUND(-99,-1,0):: XX=SGN(RND-.5)
1210 FOR KO=1 TO 40 STEP 3 :: CALL MOTION(#1,0,KO*XX):: CALL COINC(#1,#2,16,KON)
:: IF KON THEN 1240
1220 CALL PATTERN(#1,96):: CALL SOUND(-200,550+(KO*3),0,660+(KO*4),0,770+(KO*5),
0):: CALL COINC(#1,#2,16,KON):: IF KON THEN 1240
1230 CALL PATTERN(#1,104):: NEXT KO
1240 CALL POSITION(#1,U3,U4):: CALL MOTION(#1,0,0):: CALL PATTERN(#2,112)
1250 FOR I=2 TO 15 :: CALL SOUND(-1000,-7,I*2):: CALL SPRITE(#I,112,INT(RND*13+3
),U3,U4):: CALL MOTION(#I,INT(RND*-2)+-1,INT(RND*3)+1*SGN(RND-.5)):: NEXT I
1260 FOR I=15 TO 2 STEP -1 :: CALL DELSPRITE(#I):: NEXT I :: GOTO 760
1270 CALL CLEAR :: END
1280 FOR I=1 TO 70 :: CALL HCHAR(INT(RND*20)+1,INT(RND*31)+1,128):: CALL HCHAR(I
NT(RND*20)+1,INT(RND*31)+1,136):: DISPLAY AT(24,6)SIZE(16):"BITTE WARTEN SIE"
1290 CALL HCHAR(INT(RND*20)+1,INT(RND*31)+1,92):: CALL HCHAR(INT(RND*20)+1,INT(R
ND*31)+1,34):: CALL HCHAR(24,8,116,16):: CALL SOUND(-88,-1,1):: NEXT I :: RETURN
1300 CALL COLOR(13,INT(RND*13)+3,1,14,INT(RND*13)+3,1,8,INT(RND*13)+3,1):: RETUR
N
1310 CALL DELSPRITE(ALL):: DISPLAY AT(8,3)SIZE(19):"PUNKTESTAND =       " :: DISP
LAY AT(12,3)SIZE(19):"BESTPUNKTE  =       " :: RETURN
1320 FOR I=1 TO 10 :: CALL COLOR(#2,7):: FOR X=1 TO 5 :: NEXT X :: CALL COLOR(#2
,11):: FOR X=1 TO 5 :: NEXT X :: CALL SOUND(-5,-1,0):: NEXT I
1330 CALL COLOR(#2,14):: DISPLAY AT(8,4)SIZE(23):"!!!KANONEN REPARATUR!!!"
1340 CALL SPRITE(#1,44,3,165,256,0,-8):: CALL SOUND(-4250,110,30,-4,5):: FOR I=1
 TO 18
1350 CALL PATTERN(#1,132):: FOR X=1 TO 10 :: NEXT X :: CALL PATTERN(#1,140):: FO
R X=1 TO 10 :: NEXT X :: CALL PATTERN(#1,44):: FOR X=1 TO 10 :: NEXT X :: NEXT I
1360 CALL MOTION(#1,0,0):: FOR X=1 TO 30 :: CALL SOUND(--20,-2,5):: NEXT X :: CA
LL COLOR(#2,11):: CALL MOTION(#1,0,-8):: CALL SOUND(-4250,110,30,-4,5)
1370 FOR I=1 TO 18
1380 CALL PATTERN(#1,132):: FOR X=1 TO 10 :: NEXT X :: CALL PATTERN(#1,140):: FO
R X=1 TO 10 :: NEXT X :: CALL PATTERN(#1,44):: FOR X=1 TO 10 :: NEXT X :: NEXT I
1390 CALL DELSPRITE(#1):: DISPLAY AT(8,3)SIZE(25):RPT$((CHR$(136)&CHR$(92)&CHR$(
34)),8):: RETURN
1400 DISPLAY AT(24,2):"JOYSTICK-TASTER 1 DRUECKEN." :: CALL KEY(1,T,S):: IF T<>1
8 THEN 1400 :: CALL CLEAR :: RETURN
1410 FOR I=1 TO TREF :: CALL SPRITE(#I,I+48,INT(RND*13)+3,80,128,INT(RND*90+1)*S
GN(RND-.5),INT(RND*90+1)*SGN(RND-.5))
1420 CALL SOUND(-99,INT(RND*1400)+330,0):: GOSUB 1300 :: NEXT I :: FOR I=1 TO 2
:: GOSUB 1300 :: NEXT I
1430 FOR I=1 TO TREF :: GOSUB 1300 :: CALL MOTION(#I,0,0):: CALL LOCATE(#I,I*17,
I*16):: NEXT I :: RETURN
1440 F=INT(RND*13)+3 :: CALL COLOR(#1,F):: RETURN
1450 DISPLAY AT(1,22)SIZE(3):USING "###":T :: RETURN
1460 CALL POSITION(#1,U3,U4):: IF U3>140 THEN 1180 :: RETURN
```



Klang aus dem Chip

Computer können nicht nur rechnen, zählen und sortieren. Die elektronischen Heinzelmännchen können weit mehr. Musik machen zum Beispiel. Heimcomputer spielen wie Klaviere, wie Saxophone oder wie Schlagzeuge. Sie spielen ein-und mehrstimmig – je nach Wunsch

Keine echte Hitparade kommt ohne sie aus: Elektronische Klangeffekte liebte schon Rock-Vater Elvis, und seine Nachfahren – von den Beatles bis zu Nina Hagen – feilten eifrig an der Musik aus Stromimpulsen. Auch so ungewöhnliche Töne wie etwa das Geheul startender Raumschiffe in „Star Wars" und das Zischen von Science-fiction-Strahlenwaffen stammen aus dem Wunderland der Elektronik.

Aufwendig und teuer ist die Erzeugung solchen Sounds in den Tonstudios früherer Tage gewesen. Die Mischpult-Regisseure fürchten nun die Konkurrenz des Home-Computers. Der macht's entschieden billiger.

Jeder, der mit dem Keyboard eines derartigen Gerätes umgehen kann, ist sein eigener Toningenieur. Auf Tastendruck lassen sich den Chips die unmöglichsten Geräusche entlocken. Aber auch den Ohren genehme Töne und ganze Musikstückchen rieseln sanft aus dem Mikrocomputer.

„Eine irre Sache", findet Ernst Huber. Der 17 Jahre alte Schüler aus München hat seinen Commodore 64 in ein piepsendes, summendes und klingendes Musikinstrument verwandelt. Per Knopfdruck wird der „Computer des Jahres 1983" plötzlich zum Synthesizer.

Das kann übrigens jeder Mikrorechner. Nur der C 64 macht's besonders schön. In einem speziellen Bauteil – dem „Sound Interface Device (SID)" – sind die Klangelemente eingepaßt. Zu dem Chip gehören drei Tongeneratoren, mischbare Filter und sogenannte Ringmodulatoren als technischer Zusatz.

Damit läßt sich schon ein ganz erhebliches Spektakel anrichten. Donner, Motorgedröhn, Schüsse und Detonationen von Sprengstoff hören sich aber erst dann so richtig

echt an, wenn eine Stereo-Anlage mit dem Computer verbunden wird. Mit einem künstlichen Donner hat Ernst auf diese Weise seine große Schwester schon mal in heillosen Schrecken versetzt. Aber auch weniger unfriedliche Geräusche können mit der Tonerzeugung des C 64 hervorgebracht werden.

Das Programm „Synthimat 64" wandelt für knapp hundert Mark die Rechner-Tastatur in ein Klavier um. Zur Hardware-Ausrüstung sind neben dem Computer ein Diskettenlaufwerk und ein Farbfernseher nötig.

Auf eine Diskette passen 256 Soundprogramme, dazu neun vollständige Lieder, die sich jeweils mit den Sounds kombinieren lassen. So sind Begleitung und Melodiestimme unabhängig voneinander zu spielen. Das wird auch auf dem Bildschirm abgebildet. Zwei verschiedene Tastaturen tauchen auf dem Monitor auf.

Das Keyboard des Homecomputers ist freilich nicht für Pianistenhände geschaffen. Martha Argerich würde es bestimmt als Zumutung von sich weisen, auf schlichten Schreibmaschinentasten ein Konzert zu geben. Profimusiker werden wohl kaum auf ein echtes Klavier verzichten wollen. Doch Hobby-Spieler und Einsteiger in die Welt des elektronischen Klangs können sich über ihren Spar-Synthesizer freuen.

Und die Funktionen sind vielfältig, die „Synthimat" der Elektronik entlockt. Beispielsweise kann eine Melodie als Begleitmusik zu einem Lied gewählt werden. Auf Druck einer bestimmten Tastenfolge spielt das Programm die Tonleiter rauf und runter. Oder schwebende Töne aus den Oszillatoren gaukeln eine Streicherbegleitung vor. Auch Akkorde klingen aus dem zum Musikinstrument erkorenen Computer.

Jede Taste, die berührt wird, wird auf dem Keyboard-Schema des Bildschirms angezeigt – wie bei einem Klavier, das von Geisterhand spielt.

Besonders einfach ist es, den ZX Spectrum als tönendes Gerät, das die Tonleiter auswendig weiß, einzusetzen. Dazu wird die „BEEP"-Taste benutzt. Die Noten müssen zuvor in Zahlen codiert werden. Dann läuft ein selbst programmiertes Musikstück ab. So bedeutet etwa 1,0 die Tonlage eines C. Die Eins legt die Länge des Tones und die zweite Zahl die Höhe fest. Für eine Sekunde lang klingt ein C auf. Aber erst mit einem Verstärker sind die Musikstücke in angenehmer Lautstärke vernehmbar.

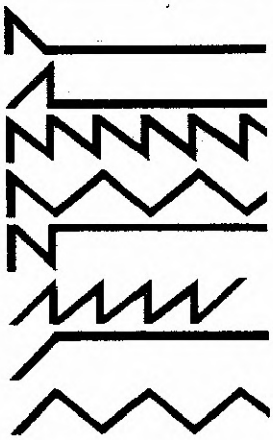
Mit einem maßgeschneiderten Programm tut der Spectrum auch mal so, als sei er ein Klavier.

Der lockere Umgang mit Musik fällt kaum einem Computer schwer.

Aber wie kommt es dazu, daß diese elektronischen Apparate überhaupt zu Komponisten und Solisten werden können? Die Elektrotechnik macht sich zunutze, daß jeder Ton – egal ob ein musikalischer oder sonst einer – aus Frequenzen besteht. Für das Ohr des Menschen ist ein Schwingungsbereich von 20 bis 15 000 Hertz hörbar. Der Computer produziert einfach Frequenzen. Aber noch andere Dinge kennzeichnen Töne. Wenn zum Beispiel eine Gitarrensaite angeschlagen wird, entsteht ein Ton, der erst laut und dann immer leiser wird. Die Anstiegszeit und die Abklingzeit beschreiben den charakteristischen Verlauf einer „Hüllkurve", die sozusagen das Profil eines Tones umreißt.

Diese Eigenheiten eines Klangs können mit Filter und Verstärker nachgeahmt werden. Im Computer sind Hüllkurvengeneratoren eingebaut, die diesen Vorgang steuern. Außerdem produziert der Computer Töne, deren Schwingungskurven wie Rechtecke, Sägezähne und Dreiecke aussehen. Sie klingen scharf, rauh oder weich. Und der Computer mischt sie. Erst ihre Mischung ergibt einen originalgetreuen Ton. Ein Synthesizer läßt programmgesteuert neben dem Grundton die entsprechenden Obertöne, die den Charakter eines bestimmten Klangs ausmachen, im richtigen Moment und mit der richtigen Stärke los. So ist es theoretisch möglich, alle existierenden Töne nachzumachen – auch mit einem Homecomputer. Bei Beethovens Neunte muß er allerdings passen. *Michael Dorfler*

Profil der Töne



Erst auf einem Oszillographen werden die Hüllkurven der Töne, die von einem Computer erzeugt werden, sichtbar. Sie sind für die Klangfarbe der Töne verantwortlich. Je nach Wahl der Hüllkurve klingt der Ton scharf, rauh oder weich

Beispiel für ein Musik-Programm

10 SI = 54272: FL = SI: FH = SI + 1: W = SI + 4: A = SI + 5: H = SI + 6: L = SI + 24	1. Definition der Registeradressen
20 POKE L, 15	2. Volle Lautstärke
30 POKE A, 16 + 9	3. Anschlag
40 POKER H, 4* 16 + 4	4. Halten und Ausklingen
50 POKE FH, 29: POKE FL, 69	5. Hi und Lo Byte der Frequenz, hier für den Kammerton A. Für andere Töne sind die beiden Werte aus der Tabelle im Anhang P zu entnehmen.
60 POKE W, 17	6. Wellenform. Muß immer als letztes eingestellt werden, da das niedrigste Bit in diesem Register den Tongenerator ein- bzw. ausschaltet.
70 FORT = 1T05000:NEXT	7. Schleife zur Einstellung der Tondauer.
80 POKE W, 0:POKE A, 0: POKE H, 0	8. Ausschalten von Wellenform- und Hüllkurveneinstellungen.

Mit freundlicher Genehmigung : Magazin "Hobby - Technik"

Noah 2099

Noha - 2099

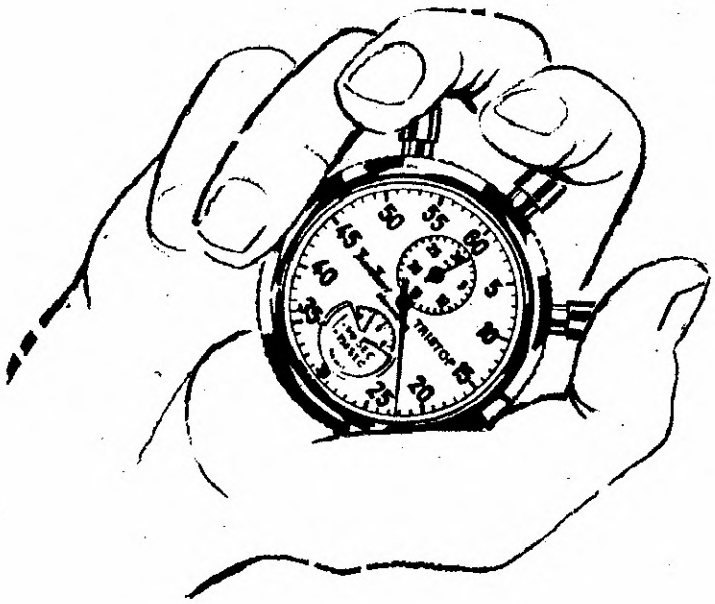
Ein Spiel in Extended Basic mit Joystick

Wir schreiben das Sternenjahr 2099. Im Sonnensystem XOR bahnt sich eine kosmische Katastrophe an, denn die Sonne bläht sich auf und schickt radioaktive Strahlung ins All, welche sämtliche Planeten des Sonnensystems XOR zu vernichten droht. Sie haben nun als NOHA die Aufgabe, sämtliche Lebewesen auf den Planeten zu evakuieren, damit diese nicht durch die Strahlung getötet werden. Steigen Sie nun ein in Ihre TI 99/4 A Arche, schieben Sie das Extended BASIC Modul in den Modulschacht und nehmen Sie den Joystick zur Hand. Es beginnt eine abenteuerliche Rettungsaktion im Sonnensystem XOR.

Das Spiel wechselt ständig zwischen zwei Spielszenen:

In der ersten Spielszene (Bild 1) fliegen Sie durch einen Meteoritenschwarm, dem es gilt auszuweichen. Steuern Sie Ihre Arche mit dem Joystick nach rechts oder links sicher durch die Meteoriten hindurch. Die Arche fliegt langsam von unten nach oben über den Bildschirm, wobei sich die Steuerung auf den Meteoritenschwarm auswirkt. Wenn die Arche den oberen Bildschirmrand erreicht hat, so wechselt das Bild blitzschnell zu Szene 2 (Bild 2) um. Die Arche senkt sich jetzt auf den Planeten herunter und verharrt etwa in Mitte des Bildschirms. Steuern Sie jetzt den Lander mittels des Joysticks aus der Arche und fliegen Sie zur Planetenoberfläche hinab, wo ein Wesen aufgeregt hin- und herläuft. Steuern Sie den Lander über das Wesen und drükken Sie den Aktionsknopf, um den Trägerstrahl auszulösen und das Wesen an Bord zu „beamen". Wenn Sie das geschafft haben, so kehren Sie mit dem Lander zur Arche zurück, wonach sich diese wieder vom Planeten entfernt und in den nächsten schnelleren Meteoritensturm hineinfliegt.

Es gibt manche Planeten, deren Athmosphäre so dicht ist, daß der Lander nicht zur Oberfläche vordringen kann.



In diesen Fällen muß die Atmosphäre mittels des ‚Beamstrahles' vernichtet werden, damit der Lander das Wesen retten kann.

Wenn der Lander nicht weit genug zur Oberfläche vordringt, so drücken Sie den Aktionsknopf. Sie sehen nun den ‚Beamstrahl'. Lassen Sie den Knopf gedrückt und bewegen Sie den Joystick nach oben, so lange, bis der ‚Beamstrahl' nicht mehr erscheint. Jetzt lassen Sie den Aktionsknopf los und drücken anschließend den Joystick nach unten. Die Atmosphäre ist jetzt dünner geworden und der Lander kann ungehindert zur Planetenoberfläche vordringen.

Für das Rettungsmanöver haben Sie selbstverständlich nur begrenzt Zeit. Wenn die Strahlung dem Planeten zu nahe kommt, so stößt die Arche ein Warnsignal aus. Jetzt heißt es, sich mit der Rettung zu beeilen, da der Lander nur nach erfolgreicher Mission zur Arche zurückkehren kann.

Das Spiel beginnt mit Runde 1 und hat seinen höchsten Schwierigkeitsgrad bei Runde 20 erreicht. Danach geht das Spiel zwar weiter, steigert jedoch die Schwierigkeit nicht mehr.

Für jeden bestandenen Meteoritensturm erhalten Sie 100mal Rundenzahl Punkte.

Jedes gerettete Wesen bringt 1000 plus 50mal Rundenzahl Punkte.

Alle 10 000 Punkte gibt es eine Bonus-Arche. Der Rechner kann maximal 15 Reservearchen bereithalten. Sie beginnen das Spiel mit 5 Archen. Die Punktanzeige geht bis 9 999 999. Danach werden Sternchen angezeigt.

Ich bin persönlich noch nicht in den Genuß gekommen, eine sechsstellige Zahl in der Punktanzeige zu sehen sowie mehr als sechs Archen mein eigen nennen zu dürfen.

Das Programm zeigt eine ungewöhnliche Programmtechnik der Sprites. So werden diese zum Teil derart überlagert, daß die Arche in zwei Farben erscheint. Weiterhin wurde sich der Unart des TI, nur max. 4 Sprites pro Zeile darstellen zu können, zu Nutze gemacht und somit das interessante ‚Verschwinden' der Arche realisiert. Soundmäßig wurde aus dem TI alles herausgeholt, was er hergibt.

Doch nun: Viel Spaß beim Spiel!

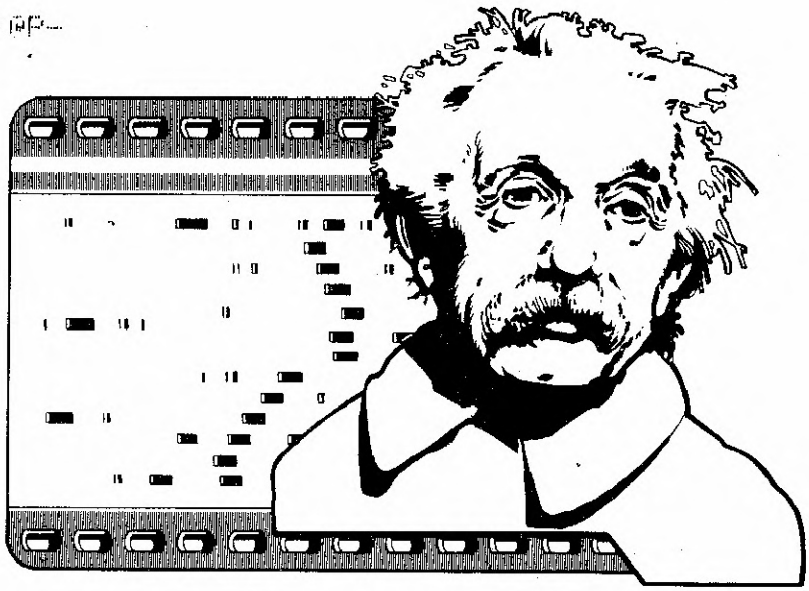
Anmerkung: Während des Spieles darf die Taste ‚ALPHA LOCK' nicht gedrückt sein, da sonst die Joysticksteuerung nicht richtig funktioniert.

```
1 !@P-
10 REM ***********************************
12 REM *        N O H A  -  2 0 9 9      *
14 REM *-----------------------------------*
16 REM * Ein Spiel fuer den TI 99/4 A     *
18 REM * geschrieben in  Extended BASIC   *
20 REM * Steuerung mittels Joystick       *
22 REM *-----------------------------------*
24 REM * (c) 1984 by Volker Becker        *
26 REM *    Steinbacher Strasse 10        *
28 REM *     6370 Oberursel 6             *
30 REM ***********************************
32 REM
34 REM
70 !@P+
72 GOTO 90 :: ARCHE :: BONUS :: I :: K :: M :: MOT :: P :: R :: RUNDE :: S :: ST
 :: STV :: T :: X :: X1 :: XV :: Y :: Y1 :: CALL ANZARCHE :: CALL ASCIISET :: CA
LL CLEAR :: CALL COINC
74 CALL COLOR :: CALL DELSPRITE :: CALL HCHAR :: CALL JOYST :: CALL KEY :: CALL
MAGNIFY :: CALL MOTION :: CALL POSITION :: CALL PUNKTE :: CALL SCREEN
76 CALL SOUND :: CALL SPRITE :: CALL STRAHLING :: CALL TITEL
80 DATA 1,2,3,4,5,6,7,8,9,0
90 OPTION BASE 1
95 DIM B(4)
96 !@P-
100 RANDOMIZE
105 CALL CLEAR :: CALL ASCIISET :: CALL TITEL(B(),5,0):: CALL COLOR(2,10,1,3,10,
1,4,10,1)
108 CALL SCREEN(2):: CALL COLOR(1,11,1,12,10,1):: CALL CLEAR
109 FOR K=33 TO 124 STEP 91
110 FOR I=1 TO 50 :: X=RND*21+1 :: Y=RND*31+1 :: CALL HCHAR(X,Y,K):: NEXT I
112 NEXT K
120 RUNDE=1 :: ARCHE=4 :: P=0 :: BONUS=1
124 CALL ANZARCHE(ARCHE):: CALL PUNKTE(P)
125 CALL MAGNIFY(4)
130 CALL SPRITE(#1,128,9,193,96,#2,132,9,193,128)
140 CALL MOTION(#1,-3,0,#2,-3,0)
150 X=RND*139+1 :: Y=X+115 :: CALL SPRITE(#10,96,4,1,X,#11,100,14,57,Y)
160 Y=RND*139+1 :: Y=X+115 :: CALL SPRITE(#12,104,11,121,X,#13,108,8,185,Y)
162 MOT=10+RUNDE
165 FOR I=10 TO 13 :: CALL MOTION(#I,MOT,5):: NEXT I
170 CALL JOYST(1,X,Y)
175 IF X=0 THEN 190
180 FOR I=10 TO 13 :: CALL MOTION(#I,MOT,-X*4):: NEXT I
190 CALL COINC(ALL,ST):: IF ST THEN GOSUB 2000 :: GOTO 130
200 CALL POSITION(#1,X,Y):: IF X<7 THEN 1000
210 CALL SOUND(-500,520-X,5,519-X,5,522-X,5,-7,5):: GOTO 170
1000 P=P+100*RUNDE :: CALL PUNKTE(P):: GOSUB 8000
1005 CALL MOTION(#1,0,0,#2,0,0):: FOR I=10 TO 13 :: CALL DELSPRITE(#I):: NEXT I
1010 CALL SCREEN(5):: CALL COLOR(1,4,1):: CALL COLOR(#1,11,#2,11):: STV=0
1020 CALL HCHAR(24,1,35,32):: CALL HCHAR(23,1,34,32)
1022 CALL POSITION(#1,X,Y):: CALL SPRITE(#3,136,9,X,Y,#4,140,9,X,Y+32)
1025 CALL MOTION(#1,3,0,#2,3,0,#3,3,0,#4,3,0)
1027 CALL SPRITE(#5,112,2,147,112)
1030 FOR I=1 TO 10
1035 MOT=-MOT :: CALL MOTION(#5,0,MOT)
1050 CALL SOUND(150,195,0):: CALL SOUND(150,261,0):: CALL SOUND(150,329,0):: CAL
L SOUND(150,523,0)
1060 NEXT I
1070 CALL MOTION(#1,0,0,#2,0,0):: CALL DELSPRITE(#3,#4)
```

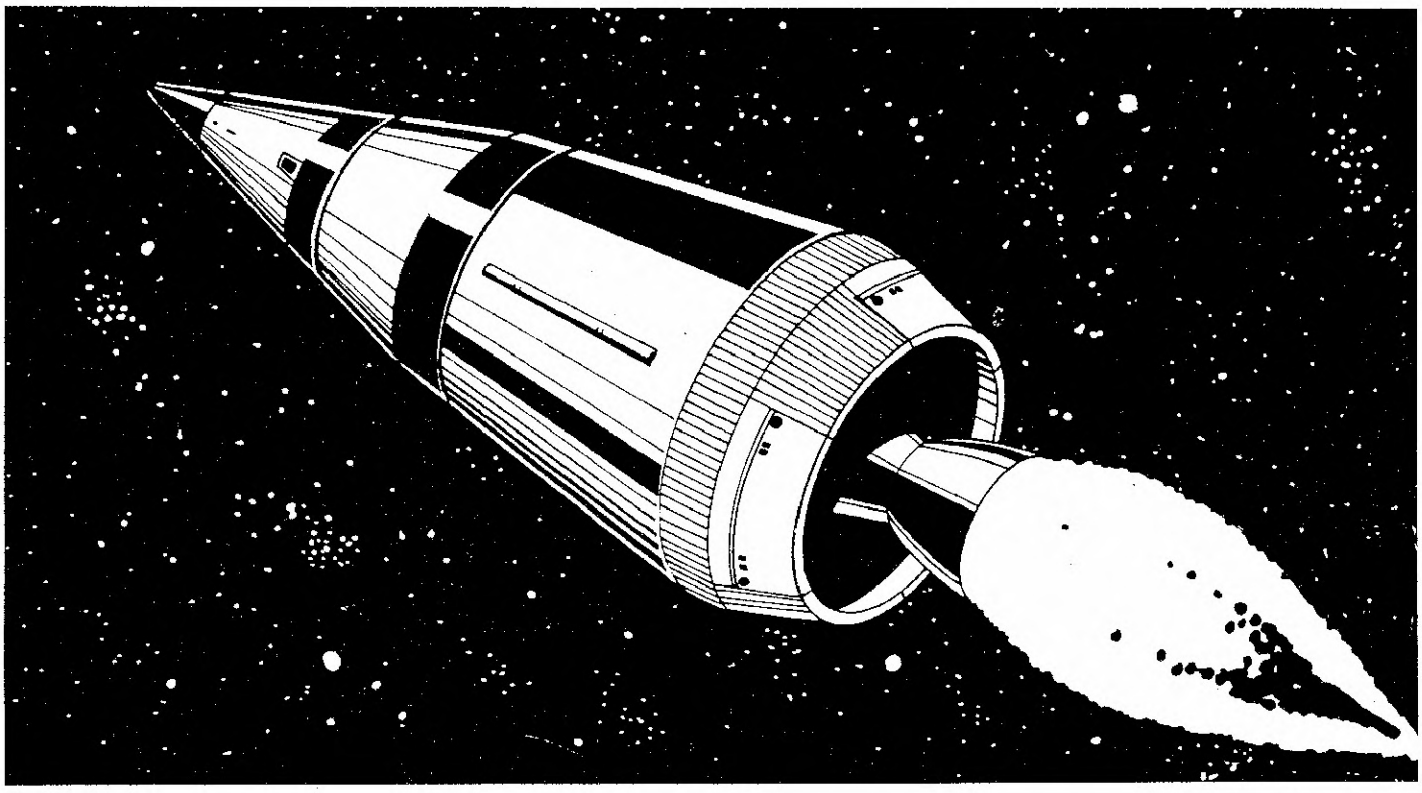
```
1080 CALL POSITION(#1,XV,Y):: CALL SPRITE(#3,116,14,XV+5,Y+16)
1090 FOR I=90-RUNDE*3 TO 0 STEP -1
1092 CALL COINC(ALL,ST):: IF ST THEN 4000
1095 M=RND*20 :: IF M>10 THEN MOT=-MOT
1100 CALL JOYST(1,X,Y):: CALL POSITION(#3,X1,Y1):: IF STV THEN 1102 ELSE IF X1<=
XV+32 THEN IF Y=4 THEN Y=0
1102 IF X1>=125 THEN IF Y=-4 THEN Y=0
1105 CALL MOTION(#3,-Y*5,X*5,#5,0,MOT):: CALL KEY(1,T,S):: IF T=18 THEN GOSUB 30
00
1107 IF I<=15 THEN CALL SOUND(-100,195,0,261,0,329,0,-7,0)ELSE CALL SOUND(-100,6
59,0)
1110 NEXT I
1112 CALL COLOR(#1,6,#2,6)
1115 CALL STRAHLUNG
1120 GOSUB 2000
1130 CALL HCHAR(23,1,32,64):: CALL SCREEN(2):: CALL COLOR(1,11,2):: GOTO 130
2000 CALL MOTION(#1,0,0,#2,0,0):: CALL DELSRRITE(#3,#5):: CALL POSITION(#1,X,Y)
2005 FOR I=10 TO 13 :: CALL DELSPRITE(#I):: NEXT I
2010 CALL SPRITE(#12,128,6,X,Y,#13,132,6,X,Y+32):: CALL SPRITE(#10,128,1,X,Y,#11
,132,1,X,Y+32)
2020 CALL SPRITE(#1,128,1,X+32,Y,0,0,#2,132,1,X+32,Y+32)
2030 CALL MOTION(#1,-5,0,#2,-5,0)
2040 FOR I=523 TO 343 STEP -10
2050 CALL SOUND(-50,I,0,I+1,0,I-1,0)
2060 NEXT I
2070 CALL DELSPRITE(ALL)
2075 ARCHE=ARCHE-1 :: CALL ANZARCHE(ARCHE):: IF ARCHE<0 THEN 10000
2080 RETURN
3000 CALL MOTION(#3,0,0):: CALL POSITION(#3,X,Y):: IF X<125 THEN RETURN ELSE CAL
L SPRITE(#4,120,12,X,Y):: CALL SOUND(-250,1046,0)
3010 CALL COINC(ALL,ST):: IF ST THEN 5000
3015 CALL DELSPRITE(#4)
3020 RETURN
4000 IF STV THEN 6000 ELSE 1095
5000 STV=-1 :: CALL DELSPRITE(#4,#5):: CALL SOUND(250,523,0):: GOTO 3020
6000 P=P+1000+50*RUNDE :: CALL PUNKTE(P):: GOSUB 8000
6005 CALL DELSPRITE(#3,#4):: CALL POSITION(#1,X,Y):: CALL SPRITE(#3,136,9,X,Y,#4
,140,9,X,Y+32)
6010 CALL MOTION(#1,-4,0,#2,-4,0,#3,-4,0,#4,-4,0)
6015 RUNDE=RUNDE+1 :: IF RUNDE>20 THEN RUNDE=20
6020 FOR I=1 TO 10
6030 CALL SOUND(150,523,0):: CALL SOUND(150,329,0):: CALL SOUND(150,261,0):: CAL
L SOUND(150,195,0)
6040 NEXT I
6050 CALL DELSPRITE(ALL)
6060 GOTO 1130
8000 IF P>=10000*BONUS THEN 8020
8010 RETURN
8020 ARCHE=ARCHE+1 :: IF ARCHE>15 THEN ARCHE=15
8030 CALL ANZARCHE(ARCHE)
8035 BONUS=BONUS+1
8040 RETURN
10000 CALL TITEL(B( ),1,1)
10010 GOTO 108
19999 !@P+
20000 SUB EINLEITUNG(B( ),KY)
20002 GOTO 20010 :: S :: T :: CALL OLMANRIVER :: !@P-
20010 CALL DELSPRITE(ALL):: CALL CLEAR :: CALL COLOR(2,2,1):: CALL SCREEN(3)
20020 DISPLAY AT(1,1):"      N O H A - 2 0 9 9          (((((((((((((((((((((((((((((("
20030 DISPLAY AT(4,1):"IM SONNENSYSTEM XOR HAT SICHEINE KOSMISCHE KATASTROPHE  E
```

```
REIGNET. DIE SONNE BLAEHT  SICH AUF UND SCHICKT RADIO-"
20040 DISPLAY AT(8,1):"AKTIVE STRAHLUNG INS ALL."
20050 DISPLAY AT(10,1):"WIR SCHREIBEN DAS JAHR 2099 SIE HABEN NUN ALS NOHA DIE
AUFGABE DIE PLANETEN DES    SONNENSYSTEMES ZU EVAKUIEREN"
20060 DISPLAY AT(14,1):"BEVOR DIE STRAHLUNG DIESE   ERREICHT, UND SOMIT SAEMT-
LICHES LEBEN AUF DEN        PLANETEN VERNICHTET."
20070 DISPLAY AT(19,1):"FLIEGEN SIE MIT IHRER ARCHE VON PLANET ZU PLANET UND
RETTEN SIE DIE WESEN, DIE   AUF DEN PLANETEN LEBEN."
20080 DISPLAY AT(24,1):"WEITER MIT  E N T E R"
20085 CALL OLMANRIVER(B(),5)
20090 CALL KEY(5,T,S):: IF S<>0 AND T=13 THEN 20100 ELSE 20085
20100 CALL CLEAR
20110 DISPLAY AT(1,1):"STEUERN SIE IHRE ARCHE MIT  DEM JOYSTICK SICHER DURCH   E
INEN METEORITENSTURM, UM   DANN EINEN PLANETEN ZU"
20120 DISPLAY AT(5,1):"ERREICHEN. DANACH STEUERN   SIE DEN LANDER AUF DIE OBER-F
LAECHE DES PLANETEN HINAB  UND BEAMEN DAS WESEN IN DEN"
20130 DISPLAY AT(9,1):"LANDER. NUN KEHREN SIE MIT  DEM LANDER ZUR ARCHE ZURUECKU
ND FLIEGEN DURCH EINEN     WEITEREN METEORITENSTURM"
20140 DISPLAY AT(13,1):"ZUM NAECHSTEN PLANETEN.     DIE METEORITENSTUERME WERDEN
IMMER SCHNELLER, DIE ZEIT,  IN WELCHER SIE DAS WESEN AN"
20150 DISPLAY AT(17,1):"BORD DES LANDERS BEAMEN     MUESSEN WIRD IMMER KUERZER
UND DAS WESEN LAEUFT IMMER  SCHNELLER HIN UND HER."
20160 DISPLAY AT(21,1):"WENN DAS WARNSIGNAL DER     ARCHE ERTOENT, MUESSEN SIE
SICH MIT DER RETTUNG BEEILENWEITER MIT  E N T E R"
20165 CALL OLMANRIVER(B(),5)
20170 CALL KEY(5,T,S):: IF S<>0 AND T=13 THEN 20180 ELSE 20165
20180 CALL CLEAR
20190 DISPLAY AT(1,1):"SIE KOENNEN NUR ZUR ARCHE   ZURUECKKEHREN, WENN SIE DAS W
ESEN GERETTET HABEN."
20200 DISPLAY AT(5,1):"STEUERUNG                   CCCCCCCCC                    D
IE ARCHE, SOWIE DER LANDER WERDEN MIT DEM JOYSTICK GE- STEUEUERT."
20210 DISPLAY AT(10,1):"ZUM AUSLOESEN DES TRAEGER-  STRAHLES WIRD DER AKTIONS-
KNOPF GEDRUECKT."
20220 DISPLAY AT(14,1):"PUNKTWERTUNG                CCCCCCCCCCCC
BESTANDENER METEORITENSTURM RUNDE MAL  100 PUNKTE"
20230 DISPLAY AT(18,1):"GERETTETES WESEN            1000 + RUNDE MAL 50 PUNKTE"
20240 DISPLAY AT(21,1):"ZU SPIELBEGINN STEHEN IHNEN 5 ARCHEN ZUR VERFUEGUNG.
BONUS ARCHE ALLE 10000 PUNKESPIELBEGINN MIT AKTIONSKNOPF"
20245 CALL OLMANRIVER(B(),1)
20250 CALL KEY(1,T,S):: IF T<>18 THEN 20245
20260 CALL CLEAR
20265 !@P+
20270 SUBEND
25000 SUB OLMANRIVER(B(),KY)
25005 GOTO 25010 :: AZ :: I :: J :: K :: M1 :: M2 :: S :: STV :: T :: !@P-
25010 DATA 1,195,261,329,391,4,329,391,1,174,220,261,349,1,261,440,2,261,523,1,2
61,440,1,195,261,329,391
25020 DATA 4,329,391,1,174,220,261,349,1,261,440,2,261,523,1,261,587,1,195,261,3
29,391,4,391,659
25030 DATA 1,174,220,261,349,1,349,587,2,349,523,1,349,587,1,195,261,329,391,2,3
91,659,2,391,783
25040 DATA 1,174,220,261,349,1,523,880,2,523,783,1,523,880,4,195,246,293,391,4,4
93,787
25050 DATA 1,493,659,2,493,587,1,493,659,4,493,783,1,493,659,2,493,587,1,493,659
25060 DATA 1,195,261,329,391,4,329,523,1,174,220,261,349,1,261,440,2,261,391,1,2
61,440
25070 DATA 2,195,261,329,391,9,329,523,0
25080 RESTORE 25010
25090 AZ,STV=0
25100 READ K :: IF K=0 THEN 25080
```

```
25110 READ B(1),B(2),B(3),B(4)
25120 FOR I=1 TO K
25130 FOR J=1 TO 4
25140 STV=STV+1 :: IF STV>AZ THEN READ AZ,M1,M2 :: STV=1
25150 CALL SOUND(400,M1,1,M2,0,B(J),5):: CALL KEY(KY,T,S):: IF S<>0 THEN SUBEXIT
25160 NEXT J :: NEXT I
25170 GOTO 25100
25175 !@P+
25180 SUBEND
28000 SUB STRAHLUNG
28005 GOTO 28010 :: C :: I :: K :: !@P-
28010 DATA 11,12,16
28020 RESTORE 28010
28030 FOR K=3 TO 1 STEP -1
28035 READ C :: CALL SCREEN(C)
28040 FOR I=10 TO 1 STEP -1
28050 CALL SOUND(-250,-7,K*I)
28060 NEXT I
28070 NEXT K
28075 !@P+
28080 SUBEND
30000 SUB ANZARCHE(ARCHE)
30005 !@P-
30010 CALL HCHAR(1,3,32,15)
30020 IF ARCHE<=0 THEN SUBEXIT
30030 CALL HCHAR(1,3,40,ARCHE)
30035 !@P+
30040 SUBEND
30050 SUB PUNKTE(P)
30055 !@P-
30060 DISPLAY AT(1,22):USING "#######":P
30065 !@P+
30070 SUBEND
31000 SUB TITEL(B(),KY,STN)
31001 GOTO 31002 :: S$ :: A$ :: I :: J :: L :: S :: T :: CALL EINLEITUNG :: CALL
 VCHAR :: !@P-
31002 CALL MAGNIFY(2)
31005 IF STN=1 THEN RESTORE 31035 :: CALL SCREEN(5):: CALL COLOR(1,11,5):: GOTO
31055
31010 CALL CLEAR :: CALL SCREEN(5):: CALL COLOR(2,11,1)
31020 CALL HCHAR(1,3,40,28):: CALL HCHAR(2,3,40,28):: CALL HCHAR(23,3,40,28):: C
ALL HCHAR(24,3,40,28)
31030 CALL VCHAR(3,3,40,20):: CALL VCHAR(3,4,40,20):: CALL VCHAR(3,29,40,20):: C
ALL VCHAR(3,30,40,20)
31035 DATA G,A,M,E,O,V,E,R
31040 DATA N,O,H,A,2,0,9,9
31045 DATA 128,130,132,134,129,131,133,135
31050 RESTORE 31040
31055 J=0
31060 FOR I=73 TO 169 STEP 32
31065 J=J+1
31070 READ S$ :: CALL SPRITE(#J,ASC(S$),4,40,I)
31080 NEXT I
31090 FOR I=73 TO 169 STEP 32
31100 J=J+1
31110 READ S$ :: CALL SPRITE(#J,ASC(S$),9,89,I)
31120 NEXT I
31121 IF STN=1 THEN 31125
31122 CALL COLOR(13,16,1)
```



```
31123 FOR L=9 TO 10 :: FOR S=15 TO 18 :: READ A$ :: CALL HCHAR(L,S,A$):: NEXT S
:: NEXT L
31124 DISPLAY AT(18,4)SIZE(20):"SPIELANLEITUNG <J/N>"
31125 DISPLAY AT(21,4)SIZE(23):"@1984 BY VOLKER BECKER"
31130 CALL OLMANRIVER(B( ),KY)
31135 IF STN=0 THEN 31160
31140 CALL CLEAR :: CALL DELSPRITE(ALL)
31150 SUBEXIT
31160 CALL KEY(5,T,S):: IF T<>74 THEN IF T<>106 THEN 31140
31170 CALL EINLEITUNG(B( ),KY)
31175 !@P+
31180 SUBEND
32000 SUB ASCIISET
32005 GOTO 32010 :: C$ :: C :: I :: CALL CHAR :: !@P-
32010 DATA 000000000001030E,070301,0000000000080C0E0,B060C08,0000000000010307,0D0
60301,0000000008080C06,F0C08,0000000001030600D,070301
32020 DATA 0000000000080C0E,70C08,0000000000010307,06030101,0000000080C060B,E0C08
,,010305030600F0206,,80C0A0C060F0406,0000000001071F0A
32030 DATA 1F01,0000000080E0F85,F88,,0000000101010101,,0000000808080808,000000281
028,,,,000001071F7FFF66,FF7F1F0701,0F3FE30FE3FFFF66,FFFFE3F1F83F0F03
32040 DATA F0FCC7F0C7FFFF66,FFFFC78F1FFCF0C,000080E0F8FEFF66,FFFEF8E08,000000010
0003F3F,3F,003E3EF83E00FFFF,FF3E3E1F0F,007C7C1F7C00FFFF
32050 DATA FF7C7CF3F,0000000800000FCFC,FC,33,000000281028,34,183C77FDFFB7FDEF,35,
F7FFFDFFBBFFFFF,40,00187F247E18,64,3C4299A1A199423C
32060 DATA 003C66666666663C,001838181818183C,003C66060C18307E,003C66061C06663C,0
00C1C2C4C7F0C0C,007F607C06066663C,001E30607C66663C
32070 DATA 007F060C1830303,003C66663C66663C,003C66663E060C38
32080 RESTORE 32010
32090 FOR I=96 TO 143 :: READ C$ :: CALL CHAR(I,C$):: NEXT I
32100 FOR I=1 TO 5 :: READ C,C$ :: CALL CHAR(C,C$):: NEXT I
32110 FOR I=48 TO 57 :: READ C$ :: CALL CHAR(I,C$):: NEXT I
32115 !@P+
32120 SUBEND
```



Jack the Digger

JACK THE DIGGER

Das Programm ist in Extended-Basic geschrieben und benötigt Joysticks

Jack's Aufgabe besteht darin, Glückssymbole aufzusammeln, die in den Stollen seiner Mine verteilt sind.

Um ein Symbol aufzunehmen, muß Jack nur darübergehen. Jedes Symbol hat einen bestimmten Wert, der vom Computer zufällig ausgewählt wird.

Haben die Symbole einen besonders hohen Wert (über 50 Punkte), wird bei deren Verteilung in den Stollen ein Tusch gespielt. Um einen besonders hohen High-Score zu bekommen, sollten gerade diese Symbole alle aufgesammelt werden. Es müssen nicht alle Symbole aufgesammelt werden. Wenn keine Möglichkeit mehr besteht, an Symbole zu kommen, dann muß Jack über die letzte Leiter in den untersten Stollen gehen. Bei dem Wort „Lift" ist eine Glocke dargestellt. Wenn Jack diese Glocke berührt, setzt sich ein Lift in Bewegung, der ihn dann in den ersten Stollen nach oben bringt, wo er seine gesammelten Punkte abliefern kann.

Für jede Runde wird dann im obersten Stollen ein Sack dargestellt. Das Spiel ist zu Ende, wenn 10 Säcke erfolgreich nach oben gebracht wurden oder wenn Jack alle Leben verloren hat.

Die Aufgabe von Jack wird dadurch erschwert, daß in den Stollen böse Geister leben. Sie bewachen die Glückssymbole. Von den Geistern darf sich Jack nicht erwischen lassen. Ebenso tödlich für ihn ist ein Berühren der Stollenwände oder ein Fehltritt auf einer Leiter. Anfang und Ende eines jeden Stollens bieten in den ersten Runden eine relative Sicherheit vor den Geistern. Absolut sicher ist Jack auf den Leitern. Sie bieten den sichersten Schutz.

Pro Runde werden die Geister schneller. Dann gilt es manchmal nur noch, das Leben von Jack zu retten (mit einem leeren Sack nach oben fahren). Verliert Jack sein Leben, muß er zur Strafe wieder von ganz oben anfangen.

Noch ein Hinweis zum Abschreiben des Programms.

Manche Zeilen werden wegen ihrer Länge nicht ganz angenommen. In diesem Fall beenden Sie die Eingabe mit „Enter". Rufen Sie diese Zeile durch gleichzeitiges Drücken von „FTCN + REDO" wieder zurück und schreiben Sie die Zeile dann komplett fertig.

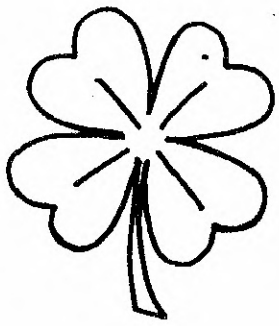
Die Graphik der Mine wird mit „FOR-NEXT"-Schleifen aufgebaut. Mit „DISPLAY AT" wäre der Aufbau schneller. Da aber die Zeichen für die Stollenwände im frei definierbaren Bereich liegen, wäre die Darstellung im Listing erschwert worden.

Und nun viel Spaß und Punkte bei diesem Spiel.

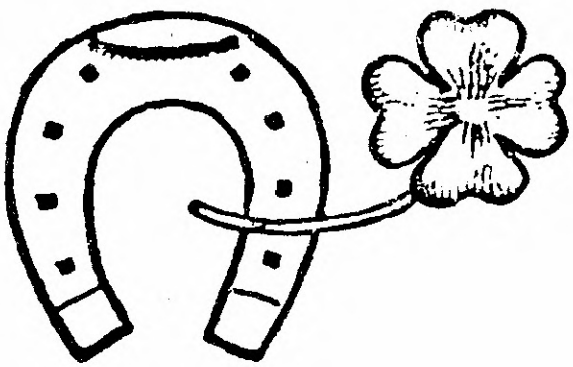
****

```
100 ! JACK THE DIGGER
110 ! WRITTEN BY W.DOELTSCH
120 ! 6108 WEITERSTADT 3
130 ! RAPPMUEHLSTR.58 C
140 ! TEL. 06150-40547
150 CALL CLEAR :: CALL MAGNIFY(2):: I=3 :: FOR T=1 TO 16 :: READ A :: CALL SPRIT
E(#T,A,5,I*16+2,I+190,0,-19):: I=I+1 :: IF A=32 THEN I=3
160 NEXT T
170 DATA 74,65,67,75,32,84,72,69,32,68,73,71,71,69,82,32
180 RANDOMIZE :: CALL SCREEN(12):: DISPLAY AT(12,3)BEEP:"ERKLAERUNGEN ??(J oder
N)"
190 CALL KEY(2,T,S):: IF T=15 THEN 230 ELSE IF T=2 THEN 200 ELSE 190
200 CALL CLEAR :: CALL SCREEN(13):: FOR I=1 TO 12 :: CALL COLOR(I,5,12):: NEXT I
210 DISPLAY AT(3,1):"***** JACK the DIGGER *****": :"Sammle alle Symbole aber":
:"beruehre nicht die Wand": :"oder die Geister."
220 DISPLAY AT(13,1):"Bringe die Punkte zum Lift": : :"10 Runden gibt's": :"Pro
Runde wird's schwerer." :: CALL TASTER
230 DISPLAY AT(10,1)BEEP ERASE ALL:"ALPHA LOCK TASTE NACH OBEN!!" :: CALL TASTER
 :: CALL SCREEN(2):: CALL COLOR(1,2,2):: FOR I=2 TO 11 :: CALL COLOR(I,16,2):: N
EXT T
```

```
240 CALL FF :: DIM M(18):: NU,TOT=0 :: EX=232 :: DEF FA=INT(RND*(14-3+1))+3
250 CALL CHAR(128,"E0F8F0FEFCF0C0E0",129,"C0F8E0FCF8C0F0E0",130,"F0F8FCFEF0F8E0F
0",131,"E0F8F0FEF8FCF8E0")
260 CALL CHAR(132,"030F071F3F0F0703",133,"071F070F1F3F0F03",134,"070F3F03071F030
7",135,"03070F1F3F0F0307")
270 CALL CHAR(136,"FFFF7F1D08",137,"FFFEFE3C3C18",138,"FFFFFE7A70202",139,"FFFFF
E74501")
280 CALL CHAR(140,"0010185A7AFFFFFF",141,"0000081A1E5FFFFF",142,"00001018 5C7DFDF
F",143,"0000001034 76F6FF")
290 CALL CHAR(124,"C0F0E0E4FCFCFFFF",125,"0707131F4F5FFFFF",126,"FFFEF8FCE4F0E0C
0",127,"FFFF7F1F3F0B0307")
300 CALL CHAR(40,"81FF81FF81FF81FF",42,"8181818181818 1FF",106,"FFFFFFFFFFFFFFFF"
,86,"0000000000000018")
310 CALL CHAR(105,"6C38106C9F825C38",41,"10107C10101038FE"):: K1$="1C1E3E7CFCAC4
6BE" :: K2$="38787C3E3F35627D" :: CALL CHAR(43,K1$)
320 CALL CHAR(112,"3034283030304 86C",113,"3030203C303020 30",114,"0C2C140C0C0C123
6",115,"0C0C043C0C0C040C")
330 CALL CHAR(116,"3838103814286 80C",117,"3838107C10282C60",118,"0000004121 12DCF
C",119,"0000001212D2DC3C")
340 CALL HCHAR(24,1,106,32):: CALL VCHAR(1,1,106,23):: CALL VCHAR(1,32,106,23)
350 FOR R=3 TO 21 STEP 3 :: FOR P1=5 TO 29 STEP 4 :: CALL HCHAR(R,P1,140):: NEXT
 P1 :: FOR P2=6 TO 30 STEP 4 :: CALL HCHAR(R,P2,141):: NEXT P2
360 FOR P3=7 TO 27 STEP 4 :: CALL HCHAR(R,P3,142):: NEXT P3 :: FOR P4=8 TO 28 ST
EP 4 :: CALL HCHAR(R,P4,143):: NEXT P4 :: NEXT R
370 FOR R=4 TO 19 STEP 3 :: FOR P1=5 TO 29 STEP 4 :: CALL HCHAR(R,P1,136):: NEXT
 P1 :: FOR P2=6 TO 30 STEP 4 :: CALL HCHAR(R,P2,137):: NEXT P2
380 FOR P3=7 TO 27 STEP 4 :: CALL HCHAR(R,P3,138):: NEXT P3 :: FOR P4=8 TO 28 ST
EP 4 :: CALL HCHAR(R,P4,139):: NEXT P4 :: NEXT R :: CALL HCHAR(22,5,32,27)
390 FOR R=2 TO 21 STEP 4 :: CALL VCHAR(R,2,128):: CALL VCHAR(R+1,2,129):: CALL V
CHAR(R+2,2,130):: CALL VCHAR(R+3,2,131):: NEXT R
400 FOR R=2 TO 21 STEP 4 :: CALL VCHAR(R,31,132):: CALL VCHAR(R+1,31,133):: CALL
 VCHAR(R+2,31,134):: CALL VCHAR(R+3,31,135):: NEXT R
410 FOR R=2 TO 29 STEP 4 :: CALL VCHAR(1,R,136):: CALL VCHAR(1,R+1,137):: CALL V
CHAR(1,R+2,138):: CALL VCHAR(1,R+3,139):: NEXT R :: CALL HCHAR(1,30,136,2)
420 FOR R=4 TO 16 STEP 3 :: CALL VCHAR(R,4,132):: CALL VCHAR(R+1,4,133):: CALL V
CHAR(R+2,4,134):: NEXT R
430 FOR R=5 TO 17 STEP 3 :: CALL VCHAR(R,5,129):: NEXT R :: FOR R=1 TO 19 STEP 3
 :: CALL VCHAR(R,31,127):: NEXT R
440 FOR R=4 TO 16 STEP 3 :: CALL VCHAR(R,5,125):: NEXT R :: FOR R=6 TO 19 STEP 3
 :: CALL VCHAR(R,5,124):: NEXT R
450 CALL HCHAR(1,2,126):: CALL HCHAR(21,2,124):: FOR R=3 TO 21 STEP 3 :: CALL VC
HAR(R,31,125):: NEXT R :: CALL HCHAR(21,3,143,2)
460 DISPLAY AT(22,1)SIZE(9)"POINTS  :" :: DISPLAY AT(22,16)SIZE(6):"LIFES:" ::
CALL HCHAR(22,25,113,5)
470 DISPLAY AT(23,1)SIZE(9)."HI-SCORE:" :: DISPLAY AT(23,16)SIZE(6):"SCORE:" ::
DISPLAY AT(20,1)SIZE(5):"LIFT+"
480 CALL DELSPRITE(ALL) :: CALL MAGNIFY(1)
490 CALL LFER :: GOSUB 890 :: GOSUB 960
500 H=32 :: V=40 :: CALL DELSPRITE(ALL)
510 CALL SPRITE(#1,112,15,32,40,#7,42,5,8,FX,#28,32,1,16,232)
520 RANDOMIZE :: ON INT(RND*20)+1 GOTO 530,540,550,560,570,580,590,600,610,620,6
30,640,650,660,670,680,690,700,710,720
530 A$="0024247E427E4281" :: GOTO 730
540 A$="6699997E247EA5A5" :: GOTO 730
550 A$="81C36699997E2418" :: GOTO 730
560 A$="6699997E245A99A5" :: GOTO 730
570 A$="42E799996666DB81" :: GOTO 730
580 A$="187EFF9999FFD899" :: GOTO 730
590 A$="003C7E99FF7E42C3" :: GOTO 730
600 A$="1C3E6B7F223E2277" :: GOTO 730
610 A$="92BAD67C6C3892FE" :: GOTO 730
620 A$="81423C5A7E1824E7" :: GOTO 730
630 A$="245A99BD3C5A6618" :: GOTO 730
```

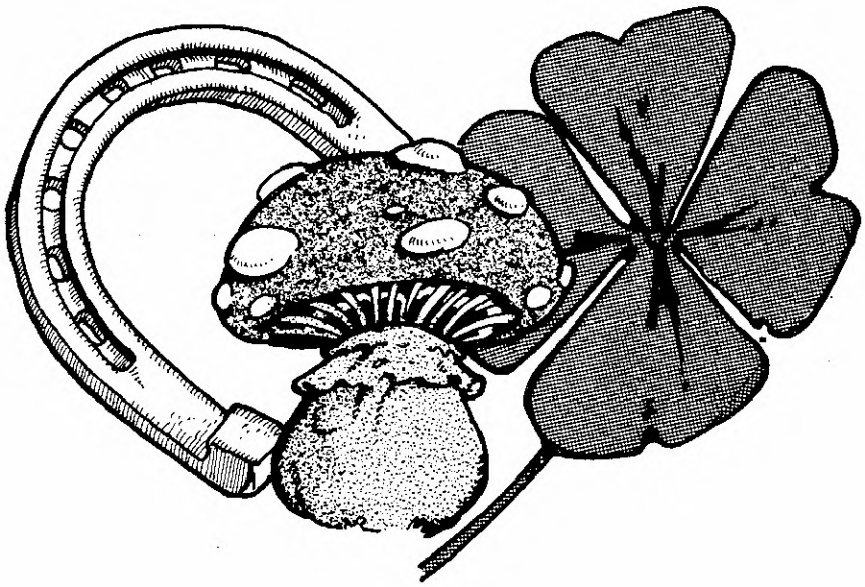


```
640 A$="66993C5A247E81E7" :: GOTO 730
650 A$="FF99FF5A0000A5FF" :: GOTO 730
660 A$="3C7E99FFFFDBA5FF" :: GOTO 730
670 A$="183C563C52492A49" :: GOTO 730
680 A$="FF9999FFE7C3C381" :: GOTO 730
690 A$="3C7EFF99FF9981FF" :: GOTO 730
700 A$="006666003C66C381" :: GOTO 730
710 A$="A5FF995A3C5A8181" :: GOTO 730
720 A$="81A55A3C243C42C3"
730 CALL CHAR(96,A$):: CALL SPRITE(#2,96,FA,32,232,#3,96,FA,56,40,#4,96,FA,80,23
2,#5,96,FA,104,40,#6,96,FA,128,232):: GOTO 1200
740 CALL COINC(ALL,HIT):: IF HIT THEN 810 :: CALL POSITION(#28,X1,X2):: IF X2<60
 THEN 1230 ELSE IF X2>200 THEN 1220
750 CALL JOYST(2,X,Y):: IF X=0 AND Y=0 THEN 740 ELSE IF X=4 THEN CALL PATTERN(#1
,112):: V=V+8 :: PA=113 :: GOTO 780
760 IF X=-4 THEN CALL PATTERN(#1,114):: V=V-8 :: PA=115 :: GOTO 780 ELSE IF Y=4
THEN CALL PATTERN(#1,116):: H=H-8 :: PA=117 :: GOTO 780
770 IF Y=-4 THEN CALL PATTERN(#1,117):: H=H+8 :: PA=116
780 CALL COINC(ALL,HIT):: IF HIT THEN 810 :: CALL LOCATE(#1,H,V):: CALL PATTERN(
#1,PA)
790 CALL COINC(ALL,HIT):: IF HIT THEN 810 :: CALL GCHAR(H/8+1,V/8+1,C):: IF C=32
 THEN 740
800 IF C=88 THEN 880 ELSE IF C>127 THEN 810 ELSE IF C=43 THEN 1240 ELSE 740
810 TOT=TOT+1 :: FL=5 :: CALL DELSPRITE(#2,#3,#4,#5,#6,#28)
820 FL=FL+3 :: CALL SOUND(-200,330+(FL*10),0,335+(FL*10),0,340+(FL*10),0)
830 CALL POSITION(#1,ZS,SS):: IF ZS<135 THEN CALL MOTION(#1,FL,0):: GOTO 820 ELS
E CALL MOTION(#1,0,0)
840 CALL LOCATE(#1,154,SS):: CALL PATTERN(#1,118)
850 FOR I=1 TO 30 STEP 2.5 :: CALL SOUND(-500,-6,I):: CALL PATTERN(#1,119):: FOR
 D=1 TO 10 :: NEXT D :: CALL PATTERN(#1,118):: FOR D=1 TO 10 :: NEXT D :: NEXT I
860 CALL DELSPRITE(#1):: CALL HCHAR(22,25,41,TOT):: IF TOT=5 THEN 1600
870 H=32 :: V=40 :: GOSUB 940 :: GOSUB 890 :: GOTO 510
880 NU=NU+PU :: DISPLAY AT(22,10)SIZE(5):NU :: CALL HCHAR((H/8)+1,(V/8)+1,32)::
GOTO 740
890 !LEITERAUFRAU
900 D=1 :: PR=0 :: FOR L=6 TO 18 STEP 3
910 M(L)=INT((RND*((20-11+1))+11)+D)
920 IF (SGN(M(L)-PR)<>D)THEN 910
930 CALL VCHAR(L,M(L),40,3):: PR=M(L):: D=-D :: NEXT L :: RETURN
940 CALL DELSPRITE(#1):: FOR L=6 TO 18 STEP 3
950 CALL VCHAR(L,M(L),INT(RND*(143-140+1))+140):: CALL VCHAR(L+1,M(L),INT(RND*(1
89-136+1))+136):: CALL VCHAR(L+2,M(L),32):: NEXT L :: CALL FF :: RETURN
960 PU=INT(RND*(100-20+1))+20 :: IF PU<50 THEN 980 ELSE CALL SOUND(250,261,9,329
,9,391,9):: CALL SOUND(250,349,12,440,12,523,12)
970 CALL SOUND(250,391,14,493,14,587,14):: CALL SOUND(4100,1046,16,783,16,329,16
)
980 ON INT(RND*17)+1 GOTO 990,1000,1010,1020,1030,1040,1050,1060,1070,1080,1090,
1100,1110,1120,1130,1140,1150
990 R$="0000206028381030" :: GOTO 1160
1000 R$="0000000008F87020" :: GOTO 1160
1010 R$="0080405264687060" :: GOTO 1160
1020 R$="0000000002448828" :: GOTO 1160
1030 R$="0000001028542810" :: GOTO 1160
1040 R$="0000101028282828" :: GOTO 1160
1050 R$="0607C0282107C8828" :: GOTO 1160
1060 R$="0000000008870888" :: GOTO 1160
1070 R$="0000407F0F0606A0" :: GOTO 1160
1080 R$="0000183C7E000000" :: GOTO 1160
1090 R$="0000103899C43810" :: GOTO 1160
1100 R$="0000001070443810" :: GOTO 1160
1110 R$="00E000007F3C24BC" :: GOTO 1160
1120 R$="0000C6652C180818" :: GOTO 1160
1130 R$="0000002828254810" :: GOTO 1160
1140 R$="0000006A2824C548" :: GOTO 1160
```



```
1150 B$="0000000C8A7C486C"
1160 CALL CHAR(88,B$):: VS,ST=0 :: FOR SYMB=5 TO 17 STEP 3 :: FOR PLATZ=1 TO 3
1170 VS=INT(RND*(30-8+1))+8 :: CALL GCHAR(SYMB,VS,ST)
1180 IF ST=32 THEN CALL HCHAR(SYMB,VS,88)ELSE 1170
1190 NEXT PLATZ :: NEXT SYMB :: RETURN
1200 S6=4 :: READ S1,S2,S3,S4,S5,S7 :: IF S7=5 THEN RESTORE 1690
1210 S1=S1+LEV :: S2=S2+LEV :: S3=S3+LEV :: S4=S4+LEV :: S5=S5+LEV :: S6=S6+LEV
1220 CALL MOTION(#28,0,-S6,#2,0,-S1,#3,0,S2,#4,0,-S3,#5,0,S4,#6,0,-S5):: GOTO 75
0
1230 CALL MOTION(#28,0,S6,#2,0,S1,#3,0,-S2,#4,0,S3,#5,0,-S4,#6,0,S5):: GOTO 750
1240 CALL DELSPRITE(#2,#3,#4,#5,#6,#28):: DISPLAY AT(20,1)SIZE(5):" "
1250 CALL HCHAR(20,4,43,3):: FOR I=1 TO 3 :: FOR A=0 TO 30 STEP 5 :: CALL SOUND(
-99,698,A,1924,A)
1260 NEXT A :: CALL CHAR(43,K2$):: FOR A=0 TO 30 STEP 5 :: CALL SOUND(-99,554,A,
1527,A):: NEXT A :: CALL CHAR(43,K1$):: NEXT I :: CALL HCHAR(20,4,32,3)
1270 FOR I=2 TO 6 :: CALL MOTION(#1,0,0):: NEXT I :: CALL SOUND(-4250,-4,0):: FO
R LI=EX TO 16 STEP -2 :: CALL LOCATE(#7,8,LI):: NEXT LI
1280 CALL SOUND(-4250,-3,9):: FOR RU=8 TO 152 STEP 2 :: CALL LOCATE(#7,RU,16):: 
NEXT RU
1290 CALL SOUND(-4250,-1,9):: FOR RE=16 TO 48 :: CALL LOCATE(#7,152,RE):: NEXT R
E
1300 CALL SOUND(-4250,-3,6):: FOR LIZ=48 TO 16 STEP -1 :: CALL LOCATE(#1,152,LIZ
,#7,152,LIZ):: NEXT LIZ
1310 CALL SOUND(-4250,-2,8):: FOR HOZ=152 TO 8 STEP -2 :: CALL LOCATE(#1,HOZ,16,
#7,HOZ,16):: NEXT HOZ
1320 CALL SOUND(-4250,-4,1):: FOR REZ=16 TO EX STEP 2 :: CALL LOCATE(#1,8,REZ,#7
,8,REZ):: NEXT REZ :: CALL DELSPRITE(#1):: CALL SOUND(-1,40000,30)
1330 SC=SC+NU :: DISPLAY AT(23,22)SIZE(5):SC :: NU=0 :: DISPLAY AT(22,10)SIZE(5)
:NU :: CALL HCHAR(2,INT(EX/8)+1,105):: EX=EX-16 :: CALL LOCATE(#7,8,EX)
1340 T=300 :: CALL SOUND(T,262,2):: CALL SOUND(T,294,2):: CALL SOUND(2*T,330,2):
: CALL SOUND(3*T/4,349,2,440,2):: CALL SOUND(T/4,392,2)
1350 CALL SOUND(T/2,440,2):: CALL SOUND(T/2,494,2,391,2):: CALL SOUND(T,523,5,39
1,4,329,0)
1360 LEV=LEV+1 :: RN=RN+1 :: IF RN=10 THEN 1380
1370 CALL LEER :: DISPLAY AT(20,1)SIZE(5):"LIFT+" :: H=32 :: V=40 :: GOSUB 940 :
: GOSUB 890 :: GOSUB 960 :: GOTO 510
1380 GOSUB 940 :: DISPLAY AT(5,6)SIZE(21):"SIE HABEN ALLE SAECKE" :: DISPLAY AT(
8,6)SIZE(23):"GUT NACH OBEN GERRACHTV"
1390 DISPLAY AT(11,6)SIZE(19):"ES KANN NOCH BESSER" :: DISPLAY AT(14,6)SIZE(20):
"WERDENV   ALSO LOSV"
1400 DISPLAY AT(17,6)SIZE(23):"PROBIEREN SIE ES BITTEV"
1410 CALL SOUND(700,4000,30):: T=450 :: CALL SOUND(T,330,2,131,6):: CALL SOUND(T
,330,3,131,6):: CALL SOUND(T,349,3,131,6)
1420 CALL SOUND(T,392,2,131,6):: CALL SOUND(T,392,3,147,6):: CALL SOUND(T,349,3,
147,6)
1430 CALL SOUND(T,330,3,147,6):: CALL SOUND(T,294,3,147,6):: CALL SOUND(T,262,2,
165,6)
1440 CALL SOUND(T,262,3,165,7):: CALL SOUND(T,294,2,175,6):: CALL SOUND(T,330,2,
176,5)
1450 CALL SOUND(T*1.5,294,2,196,7):: CALL SOUND(T/2,262,3,196,7):: CALL SOUND(4*
T,262,4,165,7,131,8):: CALL EE :: CALL EE
1460 IF SC<=HSC THEN 1620
1470 HSC=SC :: CALL SOUND(800,4444,30):: DISPLAY AT(23,10)SIZE(5):HSC
1480 T=500 :: U=T*3/4 :: CALL DELSPRITE(#7)
1490 CALL LEER :: CALL SOUND(1,9999,30):: CALL SOUND(U,156,6):: DISPLAY AT(5,6)S
IZE(24):"DAS IST EINE NEUE" :: CALL SOUND(T/4,208,5)
1500 CALL SOUND(T,262,3):: CALL SOUND(U,262,4):: DISPLAY AT(8,6)SIZE(24):"BESTLE
ISTUNGV DAFUER" :: CALL SOUND(T/4,233,4)
1510 CALL SOUND(U,208,3):: DISPLAY AT(11,6)SIZE(20):"EINE SONDERMELODIEV" :: CAL
L SOUND(T/4,262,4):: CALL SOUND(T+U,175,3,139,10)
1520 DISPLAY AT(14,6)SIZE(24):"MACHEN SIE WEITER SOV" :: CALL SOUND(T/4,220,3):: 
 CALL SOUND(U,233,3):: CALL SOUND(T/4,262,2):: CALL SOUND(U,277,2,233,8)
1530 DISPLAY AT(17,6)SIZE(24):"STEIGERN SIE SICH" :: CALL SOUND(T/4,262,3):: CAL
```

```
1 SOUND(U,233,2,196,8):: CALL LEER :: CALL SOUND(T/4,277,2)
1540 DISPLAY AT(5,6)SIZE(24):"ES GEHT GLEICH WEITER" :: CALL SOUND(U,196,2,156,8
):: CALL SOUND(T/4,233,2):: CALL SOUND(U,165,3,131,8)
1550 DISPLAY AT(8,6)SIZE(24):"MIT EINER NEUEN RUNDEV"
1560 CALL SOUND(T/4,196,3):: CALL SOUND(T+U,262,2):: CALL SOUND(T/4,233,4):: CAL
L SOUND(U,220,4,175,10)
1570 DISPLAY AT(11,6)SIZE(24):"ABER BITTE NUR DANN"
1580 CALL SOUND(T/4,175,4):: CALL SOUND(U,233,2,117,10):: CALL SOUND(T/4,277,3):
: DISPLAY AT(14,6)SIZE(24):"WENN SIE DAS WOLLENV"
1590 CALL SOUND(U,196,4,156,10):: CALL SOUND(T/4,156,4):: CALL SOUND(2*T,208,2):
: DISPLAY AT(17,6)SIZE(24):"ES LIEGT NUR AN IHNENV" :: GOTO 1630
1600 GOSUB 940 :: DISPLAY AT(5,7)SIZE(19):"LEIDER HAT ES NICHT" :: DISPLAY AT(8,
7)SIZE(9):"GEKLAPPTV" :: DISPLAY AT(11,7)SIZE(20):"LASSEN SIE BITTE DEN"
1610 DISPLAY AT(14,7)SIZE(22):"KOPF'NICHT HAENGEN UND" :: DISPLAY AT(17,7)SIZE(2
1):"WAGEN SIE ES NOCHMALV"
1620 DISPLAY AT(23,10)SIZE(5):NSC
1630 FOR I=2 TO 5 :: CALL SPRITE(#I,96,FA,152,256,0,INT(RND*35+8)):: NEXT I
1640 DISPLAY AT(20,7)SIZE(20):"NEUES SPIEL J / N ?" :: CALL FF
1650 CALL KEY(3,T,S):: IF T=78 THEN END ELSE IF T=74 THEN 1660 ELSE DISPLAY AT(2
0,7)SIZE(20):RPT$(CHR$(88)&CHR$(32),10):: CALL FF :: GOTO 1640
1660 LEV,NU,TOT,SC,RN=0 :: FX=232 :: CALL LEER :: DISPLAY AT(22,10)SIZE(5):NU ::
 DISPLAY AT(23,23)SIZE(5):SC
1670 CALL HCHAR(2,5,32,26):: CALL HCHAR(22,25,113,5):: GOSUB 890 :: GOSUB 1160 :
: DISPLAY AT(20,1)SIZE(5):"LIFT+" :: GOTO 500
1680 DATA 2,4,4,2,3,0,2,2,4,4,3,0,3,4,3,2,2,0,2,2,3,3,4,0,3,3,4,3,4,0,2,3,3,3,4,
0
1690 DATA 2,4,3,3,4,0,2,3,3,4,2,0,2,4,2,4,2,0,3,2,1,4,3,0,2,4,2,3,4,0,2,4,3,2,4,
5
1700 SUB FF :: FB=INT(RND*(14-3+1))+3
1710 FC=INT(RND*(14-3+1))+3 :: IF FC=FB THEN 1710
1720 FD=INT(RND*(14-3+1))+3 :: IF FD=FC OR FD=FB THEN 1720
1730 FE=INT(RND*(14-3+1))+3 :: IF FE=FD OR FE=FC OR FE=FB THEN 1730
1740 FG=INT(RND*(14-3+1))+3 :: IF FG=FC OR FG=FB OR FG=FE OR FG=FD THEN 1740
1750 CALL COLOR(12,FC,2,13,FC,2,14,FB,2,2,FE,2,10,FD,2,8,FG,2):: SUBEND
1760 SUB TASTER
1770 DISPLAY AT(24,2):"JOYSTICK-TASTER 2 DRUECKEN"
1780 CALL KEY(2,T,S):: IF S=0 OR T<>18 THEN 1770
1790 CALL CLEAR :: SUBEND
1800 SUB LEER :: FOR I=5 TO 20 STEP 3 :: CALL HCHAR(I,8,32,23):: NEXT I :: SUBEN
D
```



Tips + Tricks

Video Controller VC 20

Im Gegensatz zu vielen anderen Computern besitzt der VC 20 einen Video-Controller. Dieser Controller übernimmt die gesamte Aufbereitung des Fernsehbildes. Da dieser Controller ab Adresse 36864 ($9000) erreichbar ist, lassen sich kaum vorstellbare Effekte erzielen. Will man beispielsweise den Bildschirm verschieben, so ist dies für den Controller keine Schwierigkeit. Mit dem Befehl POKE 36864,X teilt man ihm die horizontale Position, und mit POKE 36865,X die vertikale Position mit. Mit diesem Trick ist es möglich, das Bild an jede Position des Schirmes zu verschieben. Hier noch eine Aufstellung sehr interessanter Adressen:

```
36864  Horizontale BildPosition (normal 12)
36865  Vertikale BildPosition (normal 38)
36866  Spaltenanzahl des Videobildes (normal 150)
36867  Zeilenanzahl des Videobildes (normal 46)
       Bit 0 waehlt 8*8 oder 16*8 Matrix der Zeichen
       Als sinnvolle Anwendung dieses Registers waere
       die Erzeugung einer feststehenden Zeile,welche sich
       nur durch POKE beschreiben laesst,denkbar.
36869  Bit 4-7 dienen zur Bestimmung der Bildschirmadresse.
       Bit 0-3 bestimmt die Lage des Zeichensatzes.
       Die folgende Tabelle zeigt den Zusammenhang der
       ersten 4 Bit's und der Position des Zeichensatzes:
```

Bit's	Adresse	Bemerkung
0000	32768	ROM Zeichensatz GROSS/GRAFIK
0010	34816	ROM Zeichensatz KLEIN/GROSS
1100	4096	Definierbarer Zeichensatz
1101	5120	" "
1110	6144	" "
1111	7168	" "

TI-99/4a

Der TI-99/4a braucht je nach Länge des Programms ca. 500-1100 Bytes für sich selbst. Wird der Platz zum internen Umdaten zu knapp, meldet er sich mit: „MEMORY VOLL IN"! . . . Mit dem Ext. Basic-Modul läßt sich das etwas ausgleichen, zudem man vor dem Einlesen des Programms folgendermaßen beginnt:

1. Befehl NEW
2. Befehl CALL FILES (1)
3. Befehl OLD CS1 oder RUN „CS1" oder
OLD DSK 1. Name oder RUN „DSK 1. Name".

TI-99/4A

Besitzt man eine Diskettenstation für den TI-99/4A ohne Speichererweiterung, dann läßt sich der CALL FILES (X)-Befehl mit in das Programm einbauen.

Zum Beispiel:

1500 REM Neues Programm anwählen
1510 CALL FILES (1)
1520 RUN „DSK 1. Name"

Je nach Menge der Daten läßt sich eine Menge Speicherplatz zusätzlich gewinnen. Mit Speichererweiterung gilt dieser Befehl als ein Unterprogramm (Programmabsturz).

News

Willkommen in der Zukunft!

Das Computerzeitalter hat begonnen und wird mehr noch als der Beginn des Industriezeitalters vor rund 200 Jahren die Gesellschaft revolutionieren.

Ob es uns gefällt oder nicht, wir alle werden mit dem Computer leben müssen, werden lernen müssen, mit ihm umzugehen.

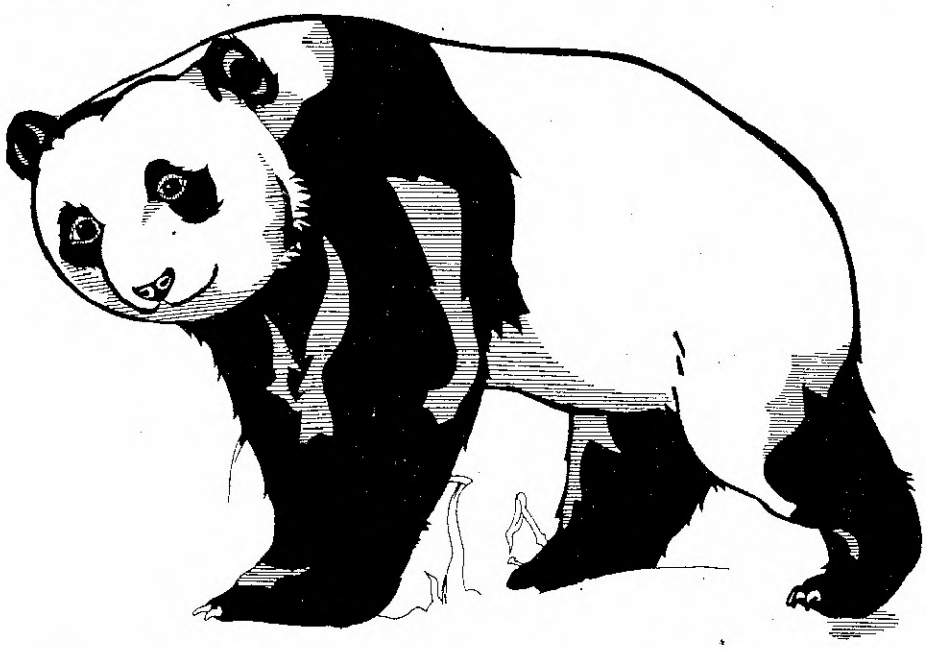
Daß auch der Technik aufgeschlossen gegenüberstehende Leute jedoch an „ihrem" Personal-Computer (Personal = persönlich) scheitern, ist kein Geheimnis. Die Ursache für Angst und Argwohn dem Computer gegenüber liegt grundsätzlich bei nicht ausreichenden Vorkenntnissen, denn auch die inzwischen so zahlreich erschienenen Computer-Bücher lassen den Anfänger in der Regel im Stich. Hier setzen die HEYNE COMPUTER-BÜCHER an. Die ersten sechs Bände (monatlich wird ab Mai ein Titel erscheinen) führen den Laien systematisch an das unbekannte Wesen Computer heran. Die HEYNE COMPUTER-BÜCHER sind von Fachleuten geschrieben, die sich über den engen Kreis der Freaks und Hacker hinaus verständlich machen können.

Das spätere Programm der HEYNE COMPUTER-BÜCHER wird einzelne Themen für den Fortgeschrittenen vertiefen.

Mit TESTEN SIE IHRE COMPUTER-INTELLIGENZ gibt Dr. Alfred W. Munzert, einer der bekanntesten Testpsychologen Amerikas, auf einfache, verständliche und fast spielerische Weise eine Einführung in Logik und Technik von Computern.

HEYNE COMPUTER-BÜCHER
Dr. Alfred W. Munzert
TESTEN SIE IHRE COMPUTER-INTELLIGENZ
Heyne-Buch Nr. 15/1, DM 9,80
Deutsche Erstausgabe

Lift Bär



Lift-Bär

Das Programm ist für den TI 99/4 A mit Extended BASIC und Joystick geschrieben.

Nach dem Eintippen, oder laden wird es mit RUN gestartet. Nach einer Pause von ca. 3 Sekunden wird der Bildschirm gelöscht. Dann belegt der Rechner den Grafiksatz um.

Der Bildschirm wird jetzt schwarz, und das Titelbild erscheint in Multicolor Großbuchstaben. Dabei wird Song of Joy gespielt.

Sie können mit der Taste „J" eine kurze Spielanleitung abrufen. Bei jeder anderen Taste erhalten Sie keine Spielanleitung.

In der Stadt wird ein Neubau errichtet. Es ist kurz nach Feierabend und die Maurer haben ihre Arbeit beendet. Sie haben jedoch einen Teil ihres Werkzeuges vergessen und einen kleinen Bären beauftragt das Werkzeug für sie zu holen. Bei dem Werkzeug handelt es sich um 2 Hämmer, 2 Gabelschlüssel und 2 Schraubendreher.

Die Aufgabe hört sich zwar leicht an, ist aber gar nicht so einfach zu lösen, da erstens der Bär nicht weiß wo das Werkzeug versteckt ist, zweitens die einzelnen Etagen des Neubaues nur mittels auf- und abfahrenden Fahrstühlen zu erreichen sind und drittens nur eine befristete Zeit zur Verfügung steht.

Sie können den Bären mit dem Joystick nach rechts und links bewegen. Achten Sie jedoch darauf, daß die Taste „ALPHA LOCK" nicht gedrückt ist, da sonst die Joystickabfrage nicht richtig funktioniert. Zum Betreten eines Liftes bewegen Sie den Bären in die Richtung, in der Sie den Lift betreten wollen, immer dann, wenn der Lift gerade vor dem Bären ist und in die gewünschte Richtung fährt.

So lange der Bär im Lift ist, und Sie nicht aussteigen wollen, muß der Joystick in der neutralen Mittelstellung stehen.

Zum Verlassen eines Liftes drücken Sie den Joystick wieder in die gewünschte Richtung.

Der Bär soll ja, wie schon zuvor erwähnt Werkzeug suchen. Das Werkzeug ist in den 12 Fragezeichen versteckt. Zum Aufheben eines Fragezeichens führen Sie den Bären genau darüber und drücken dann den Aktionsknopf am Joystick.

Wenn sich unter dem Fragezeichen ein Werkzeug befindet so ertönt ein Multiklang Gong und das Werkzeug erscheint in der oberen linken Bildschirmecke. Befindet sich unter dem aufgenommenen Fragezeichen jedoch kein Werkzeug, so werden Sie ein Kratzen vernehmen und es kann sein, daß die Wände des Neubaues versetzt werden. Dies kann Ihnen einen günstigeren Weg verschaffen, es kann Ihnen jedoch auch den Weg versperren, so daß Sie einen Umweg zum nächsten Fragezeichen wählen müssen.

Wenn Sie alle 6 Werkzeuge gefunden haben, so bekommen Sie zu Ihren Punkten den verbliebenen Zeitbonus gutgeschrieben, und es geht mit der nächsten Runde weiter.

Insgesamt wird der Schwierigkeitsgrad 7 mal erhöht. Danach bleibt die Schwierigkeit konstant.

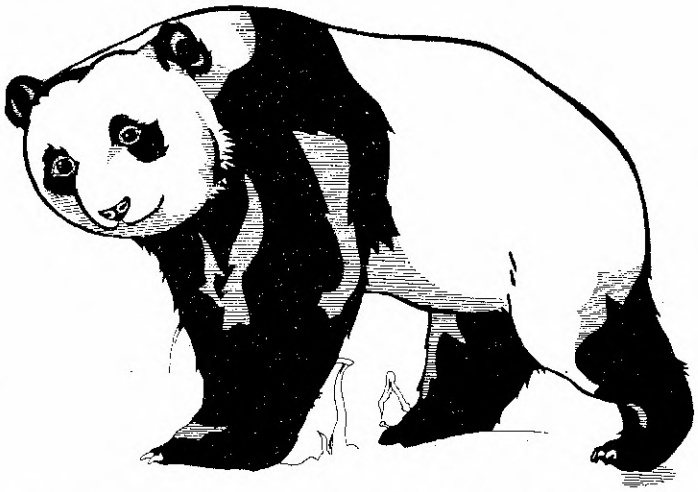
Für jeden gefundenen Hammer bekommen Sie 80 Punkte, der Gabelschlüssel bringt 100 Punkte und der Schraubendreher sogar 120 Punkte.

Für alle 5000 erreichten Punkte erhalten Sie ein Bären-Bonusleben gutgeschrieben. Der Bär verliert ein Leben wenn er

1.) zu hoch oder zu tief aus dem Lift springt.
2.) gegen eine vertikale Mauer läuft.
3.) mit dem Lift versucht durch eine Mauer zu fahren.
4.) den Bildschirm nach irgend einer Seite verläßt.
5.) wenn der Zeitbonus auf 0 abgelaufen ist.

Die Fahrstühle fahren immer auf und ab. Bei jedem Richtungswechsel wird der anfänglich 3500 Punkte zählende Zeitbonus um 50 Punkte verringert. Mit steigendem Schwierigkeitsgrad fahren die Lifte immer schneller auf und ab.

Somit läuft auch der Zeitbonus immer schneller, und das Ein- und Aussteigen wird immer schwieriger. Nachfolgend ist das Programm in groben Zügen beschrieben. Den Exakten und strukturierten Programmaufbau können erfahrene Programmierer der beigefügten Zuordnungstafel entnehmen.



```
100 !@P-
110 REM ****************************
120 REM *    L I F T  -  B A E R    *
130 REM *----------------------------*
140 REM * Ein Spiel fuer TI 99/4 A  *
150 REM * in TI Extended BASIC      *
160 REM * mit Joystick zu steuern   *
170 REM *----------------------------*
180 REM * (c) 1984 by Volker Becker *
190 REM *     Steinbacher Str. 10   *
200 REM *     6370 Oberursel 6      *
210 REM ****************************
220 REM
230 REM
240 !@P+
250 OPTION BASE 1
260 DIM VFELD(6),XFELD(9),SF(4),WFELD(3)
270 GOTO 310 :: A :: B :: BER :: BONUS :: BV :: C :: F :: I :: K :: MOT :: P ::
RUNDE :: S :: ST :: SVAR :: T :: V :: VWERT :: X :: X1 :: X2 :: Y :: Z
280 CALL ASCIISET :: CALL BAER :: CALL CLEAR :: CALL COINC :: CALL COLOR :: CALL
 DELSPRITE :: CALL EINLEITUNG :: CALL GCHAR :: CALL GONG :: CALL HCHAR
290 CALL JOYST :: CALL KEY :: CALL LOCATE :: CALL MAGNIFY :: CALL MELODY :: CALL
 MOTION :: CALL PATTERN :: CALL POSITION :: CALL PUNKTE :: CALL SCREEN
300 CALL SONGOFJOY :: CALL SOUND :: CALL SPRITE :: CALL TITEL :: CALL VCHAR :: C
ALL ZEIT :: !@P-
310 CALL CLEAR :: CALL COLOR(5,1,16,6,1,16,7,1,16,8,1,16):: CALL COLOR(3,16,1,4,
16,1,2,14,1,9,6,16)
320 RANDOMIZE
330 CALL CLEAR :: CALL ASCIISET
340 CALL CLEAR :: CALL TITEL(XFELD(),5,SF(),0,2):: CALL CLEAR
350 CALL KEY(5,T,S):: IF T=74 OR T=106 THEN CALL EINLEITUNG
360 CALL CLEAR
370 CALL SCREEN(2)
380 FOR I=4 TO 20 STEP 4
390 CALL HCHAR(I,4,112,12):: CALL HCHAR(I,18,112,12)
400 NEXT I
410 CALL HCHAR(24,1,113,32)
420 CALL VCHAR(1,1,112,24):: CALL VCHAR(1,32,112,24)
430 !@P+
440 DATA 1,8,120,1,12,128,5,6,128,5,11,136,5,23,120,5,27,136,9,4,120,9,29,128
450 !@P-
460 DATA 13,9,128,13,13,136,13,18,120,13,24,136,17,12,120,17,22,128,17,27,136,21
,4,136,21,15,128,21,28,120
470 DATA 8,2,120,8,16,120,8,30,136,12,2,128,12,16,136,12,30,128,16,2,136,16,16,1
28,20,30,120
480 RESTORE 440
490 FOR I=1 TO 18 :: READ A,B,C :: CALL VCHAR(A,B,C,3):: NEXT I
500 FOR I=1 TO 9 :: READ A,B,C :: CALL HCHAR(A,B,C,2):: NEXT I
510 CALL COLOR(11,7,2,13,7,2,12,1,1,14,1,1,9,6,16):: CALL SCREEN(5)
520 RUNDE=1 :: BER=4 :: Z=3500 :: P=0 :: BONUS=1 :: CALL PUNKTE(P):: CALL ZEIT(Z
):: CALL BAER(BER)
530 CALL MAGNIFY(3)
540 CALL SPRITE(#2,116,13,177,9,#3,116,13,1,121,#4,116,13,177,233)
550 MOT=RUNDE+4 :: CALL MOTION(#2,-MOT,0,#3,MOT,0,#4,-MOT,0)
560 DATA 3,10,3,26,7,9,7,25,11,13,11,23,15,11,15,21,19,8,19,25,23,13,23,22
570 GOSUB 820
580 BV=128
590 CALL SPRITE(#1,124,11,169,41)
600 ON ERROR 730
610 CALL ZEIT(Z)
620 REM Hauptschleife
630 V=V+1 :: CALL POSITION(#3,X,Y):: IF X>200 THEN GOSUB 940
640 CALL JOYST(1,X,Y):: IF X=0 THEN 660 ELSE IF X=-4 THEN CALL PATTERN(#1,124)EL
SE CALL PATTERN(#1,132)
650 CALL SOUND(-5,523,0)
```

```
660 CALL MOTION(#1,0,2*X)
670 CALL COINC(ALL,ST):: IF ST THEN 970
680 CALL POSITION(#1,X,Y):: X=(X+7)/8+2 :: Y=(Y)/8+1 :: CALL GCHAR(X,Y,X1)
690 IF X1=BV OR X1=112 OR X1=113 THEN 700 ELSE 730
700 CALL GCHAR(X-2,Y,X1):: CALL GCHAR(X-2,Y+1,X2):: IF X1=BV OR X2=BV THEN 730
710 CALL KEY(1,T,S):: IF T=18 THEN 1080
720 GOTO 620
730 FOR I=1 TO 4 :: CALL MOTION(#I,0,0):: NEXT I
740 CALL PATTERN(#1,140)
750 CALL MELODY
760 BER=BER-1 :: CALL BAER(BER)
770 IF BER<0 THEN CALL DELSPRITE(ALL):: BER=0 :: CALL BAER(BER):: CALL TITEL(XFE
LD(),1,SF(),-1,5):: GOTO 370
780 IF SVAR THEN Z=3500 :: SVAR=0 :: GOTO 1370
790 CALL MOTION(#2,-MOT,0,#3,MOT,0,#4,-MOT,0)
800 SVAR=0
810 GOTO 590
820 RESTORE 560
830 FOR I=1 TO 6
840 VFELD(I)=2
850 NEXT I
860 FOR I=1 TO 12
870 READ A,B
880 F=INT(RND*5+1.4):: IF VFELD(F)=0 THEN 880
890 VFELD(F)=VFELD(F)-1
900 CALL HCHAR(A,B,F+95)
910 NEXT I
920 FOR K=1 TO 3 :: WFELD(K)=0 :: NEXT K
930 RETURN
940 IF V<5 THEN RETURN ELSE V=0 :: MOT=-MOT :: CALL MOTION(#2,-MOT,0,#3,MOT,0,#4
,-MOT,0)
950 Z=Z-50 :: CALL ZEIT(Z):: IF Z<=0 THEN SVAR=-1 :: GOTO 730
960 RETURN
970 FOR I=2 TO 4
980 CALL COINC(#1,#I,16,ST):: IF ST THEN 1010
990 NEXT I
1000 GOTO 720
1010 CALL POSITION(#I,X,Y):: CALL LOCATE(#1,X,Y)
1020 IF I/2-INT(I/2)=0 THEN CALL MOTION(#1,-MOT,0)ELSE CALL MOTION(#1,MOT,0)
1030 CALL POSITION(#1,X,Y):: CALL SOUND(-250,523-X,3):: X=X/8+1 :: Y=Y/8+1 :: CA
LL GCHAR(X,Y,X1):: CALL GCHAR(X,Y+1,X2):: IF X1=BV OR X2=BV THEN 730
1040 CALL JOYST(1,X,Y):: IF X=0 THEN 1030
1050 CALL POSITION(#1,X1,Y)
1060 IF X=-4 THEN CALL LOCATE(#1,X1,Y-18)ELSE CALL LOCATE(#1,X1,Y+18)
1070 GOTO 720
1080 FOR I=1 TO 4 :: CALL MOTION(#I,0,0):: NEXT I
1090 VWERT=0
1100 CALL POSITION(#1,X,Y):: X=(X+7)/8+1 :: Y=(Y)/8+1
1110 CALL GCHAR(X,Y,X1):: CALL GCHAR(X,Y+1,X2)
1120 IF X1<102 AND X1>95 THEN VWERT=X1
1130 IF X2<102 AND X2>95 THEN VWERT=X2
1140 IF VWERT=0 THEN 1170
1150 VWERT=VWERT-95
1160 ON VWERT GOSUB 1190,1210,1220,1230,1240,1250
1170 CALL MOTION(#2,-MOT,0,#3,MOT,0,#4,-MOT,0)
1180 GOTO 720
1190 CALL HCHAR(X,Y,32,2)
1200 CALL COLOR(13,7,2,12,1,1,14,1,1):: CALL SOUND(-500,-7,0):: BV=128 :: RETURN
1210 CALL HCHAR(X,Y,32,2):: CALL COLOR(12,7,2,13,1,1,14,1,1):: CALL SOUND(-500,-
7,0):: BV=120 :: RETURN
1220 CALL HCHAR(X,Y,32,2):: CALL COLOR(14,7,2,13,1,1,12,1,1):: CALL SOUND(-500,-
7,0):: BV=136 :: RETURN
1230 CALL HCHAR(X,Y,32,2):: WFELD(1)=WFELD(1)+1 :: GOSUB 1260 :: RETURN
1240 CALL HCHAR(X,Y,32,2):: WFELD(2)=WFELD(2)+1 :: GOSUB 1260 :: RETURN
1250 CALL HCHAR(X,Y,32,2):: WFELD(3)=WFELD(3)+1 :: GOSUB 1260 :: RETURN
1260 CALL HCHAR(WFELD(VWERT-3),VWERT,VWERT+36)
```

```
1270 P=P+20*VWERT :: CALL PUNKTE(P)
1280 IF BONUS*5000<=P THEN BER=BER+1 :: CALL BAER(BER):: BONUS=BONUS+1
1290 CALL GONG
1300 FOR I=1 TO 3
1310 IF WFELD(I)<2 THEN RETURN
1320 NEXT I
1330 RUNDE=RUNDE+1 :: IF RUNDE>8 THEN RUNDE=8
1340 P=P+Z :: CALL PUNKTE(P):: CALL ZEIT(0):: Z=3500
1350 IF BONUS*5000<=P THEN BER=BER+1 :: CALL BAER(BER):: BONUS=BONUS+1
1360 CALL SONGOFJOY(SF())
1370 CALL DELSPRITE(#1):: CALL HCHAR(1,4,32,3):: CALL HCHAR(2,4,32,3)
1380 CALL COLOR(13,7,2,12,1,1,14,1,1)
1390 GOTO 540
1400 !@P+
1410 SUB SONGOFJOY(B())
1420 GOTO 1430 :: AZ :: I :: J :: K :: M1 :: M2 :: STV :: !@P-
1430 DATA 1,195,246,293,391,2,391,493,1,440,523,1,493,587
1440 DATA 1,184,261,293,369,1,493,587,1,440,523,1,391,493,1,329,440
1450 DATA 1,195,246,293,391,2,246,391,1,261,440,1,293,493
1460 DATA 1,184,261,293,369,2,277,493,2,261,440
1470 DATA 1,195,246,293,391,2,391,493,1,440,523,1,493,587
1480 DATA 1,184,261,293,369,1,493,587,1,440,523,1,391,493,1,329,440
1490 DATA 1,195,246,293,391,2,246,391,1,261,440,1,293,493
1500 DATA 1,184,261,293,369,4,261,440,1,195,246,293,391,4,246,391,0
1510 RESTORE 1430
1520 AZ,STV=0
1530 READ K :: IF K=0 THEN SUBEXIT
1540 READ B(1),B(2),B(3),B(4)
1550 FOR I=1 TO K
1560 FOR J=1 TO 4
1570 STV=STV+1 :: IF STV>AZ THEN READ AZ,M1,M2 :: STV=1
1580 CALL SOUND(400,M1,1,M2,0,B(J),5)
1590 NEXT J :: NEXT I
1600 GOTO 1530
1610 !@P+
1620 SUBEND
1630 SUB TITEL(A(),KY,B(),SVAR,FARBE)
1640 GOTO 1650 :: AZ :: I :: J :: K :: M1 :: M2 :: S :: STV :: T :: V :: SPI$ ::
 !@P-
1650 CALL MAGNIFY(2):: IF SVAR=0 THEN CALL CLEAR :: CALL SCREEN(2)
1660 DATA L,I,F,T,B,A,E,R
1670 DATA G,A,M,E,O,V,E,R
1680 CALL HCHAR(6,13,114,8):: CALL HCHAR(7,13,122,8):: CALL HCHAR(8,13,129,8)::
CALL HCHAR(9,13,137,8)
1690 RESTORE 1660
1700 IF SVAR THEN RESTORE 1670
1710 IF SVAR=0 THEN DISPLAY AT(18,3):"SPIELANLEITUNG ? <J : N>"
1720 J=1
1730 FOR K=6 TO 8 STEP 2
1740 FOR I=13 TO 19 STEP 2
1750 READ SPI$ :: CALL SPRITE(#J,ASC(SPI$),FARBE,(K-1)*8+1,(I-1)*8+1)
1760 J=J+1
1770 NEXT I
1780 NEXT K
1790 DISPLAY AT(24,3)SIZE(23):"© 1984 BY VOLKER BECKER"
1800 DATA 7,10,11,12,3,13,8,14,5
1810 RESTORE 1800
1820 FOR I=1 TO 9
1830 READ A(I)
1840 NEXT I
1850 CALL COLOR(11,A(1),A(2),12,A(3),A(4),13,A(5),A(6),14,A(7),A(8))
1860 RESTORE 1430
1870 AZ,STV=0
1880 READ K :: IF K=0 THEN 1860
1890 READ B(1),B(2),B(3),B(4)
1900 FOR I=1 TO K
```

```
1910 FOR J=1 TO 4
1920 STV=STV+1 :: IF STV>AZ THEN READ AZ,M1,M2 :: STV=1
1930 CALL SOUND(600,M1,1,M2,0,B(J),5):: CALL KEY(KY,T,S):: IF S<>0 THEN 1980
1940 FOR V=9 TO 2 STEP -1 :: A(V)=A(V-1):: NEXT V :: A(1)=A(9)
1950 CALL COLOR(11,A(1),A(2),12,A(3),A(4),13,A(5),A(6),14,A(7),A(8))
1960 NEXT J :: NEXT I
1970 GOTO 1880
1980 CALL DELSPRITE(ALL):: CALL CLEAR
1990 CALL COLOR(11,1,1,12,1,1,13,1,1,14,1,1)
2000 !@P+
2010 SUBEND
2020 SUB PUNKTE(P)
2030 !@P-
2040 DISPLAY AT(1,23)SIZE(5):USING "#####":P :: CALL HCHAR(1,24,38)
2050 !@P+
2060 SUBEND
2070 SUB ZEIT(Z)
2080 !@P-
2090 DISPLAY AT(1,17)SIZE(4):USING "####":Z :: CALL HCHAR(1,18,35)
2100 !@P+
2110 SUBEND
2120 SUB BAER(BER)
2130 !@P-
2140 DISPLAY AT(1,12)SIZE(2):USING "##":BER :: CALL HCHAR(1,13,36)
2150 !@P+
2160 SUBEND
2170 SUB MELODY
2180 GOTO 2190 :: A :: I :: L :: !@P-
2190 DATA 750,195,250,220,500,233,500,195,1000,277
2200 RESTORE 2190
2210 FOR I=1 TO 5 :: READ L,A :: CALL SOUND(L,A,0):: NEXT I
2220 !@P+
2230 SUBEND
2240 SUB GONG
2250 GOTO 2260 :: I :: !@P-
2260 FOR I=0 TO 9 :: CALL SOUND(-150,329,I):: NEXT I
2270 FOR I=0 TO 9 :: CALL SOUND(-150,277,I,329,I+11):: NEXT I
2280 FOR I=0 TO 9 :: CALL SOUND(-150,220,I,277,I+11,329,I+21):: NEXT I
2290 !@P+
2300 SUBEND
2310 SUB EINLEITUNG
2320 GOTO 2330 :: S :: T :: !@P-
2330 CALL CLEAR :: CALL SCREEN(5)
2340 DISPLAY AT(1,1):"DIE MAURER HABEN VERGESSEN  IHR WERKZEUG EIZUSAMMELN UNDDI
ESE AUFGABE MUSS JETZT    DER LIFT BAER UEBERNEHMEN."
2350 DISPLAY AT(5,1):"DAS WERKZEUG IST IN DEN a   VERSTECKT. JE NACH WERKZEUG GI
BT ES :"
2360 DISPLAY AT(8,1):"( =  80 PUNKTE                ) = 100 PUNKTE              *
= 120 PUNKTE"
2370 DISPLAY AT(11,1):"JEDES WERKZEUG IST 2 MAL    VORHANDEN. WIRD JEDOCH EIN  a
 AUFGENOMMEN IN WELCHEM    KEIN WERKZEUG IST SO WERDEN"
2380 DISPLAY AT(15,1):"DIE WAENDE DES NEUBAUES     VERSETZT."
2390 DISPLAY AT(17,1):"LIFT BAER VERLIERT EIN LEBENWENN ER IM FALSCHEN MOMENT  A
US DEM LIFT SPRINGT, EINE  MAUER BERUEHRT ODER DER"
2400 DISPLAY AT(21,1):"ZEITBONUS ABGELAUFEN IST.   BONUSLEBEN ALLE 5000 PUNKTE.N
EUES SPIEL MIT AKTIONSKNOPF"
2410 CALL KEY(1,T,S):: IF T=18 THEN SUBEXIT ELSE 2410
2420 !@P+
2430 SUBEND
2440 SUB ASCIISET
2450 GOTO 2460 :: A$ :: I :: CALL CHAR :: !@P-
2460 DATA 003C66666666663C,001838181818183C,003C66060C18307E,003C66061C06663C,00
0C1C2C4C7E0C0C,007E607C0606663C,001C30607C666 63C
2470 DATA 007E060C1830303,003C66663C66663C,003C66663E060C38,C3BD665E5E66BDC3,FFC
3999981999999,FF83C9C9C3C9C983,FFC3999F9F9F99C3
```

```
2480 DATA FF83C9C9C9C9C983,FF819F9F839F9F81,FF819F9F839F9F9F,FFC19F9F919999C3,FF
99999981999999,FFC3E7E7E7E7E7C3,FFF9F9F9F9F999C3
2490 DATA FF9993878F879399,FF9F9F9F9F9F9F81,FFBD9981A5999999,FF99898981919199,FF
81999999999981,FF839999839F9F9F,FFC399999999593C9
2500 DATA FF83999983879399,FFC3999FC3F999C3,FF81E7E7E7E7E7E7,FF999999999999C3,FF
999999C3C3E7E7,FF999999A58199DB,FF99DBC3E7C3DB99
2510 DATA FF99DBC3E7E7E7E7,FF81F9F3E7CF9F81,FEFEFE00EFEFEF,FF18FF18FF18FF18,FFFF
FFFF,0000003C40380478,1F306FD0AFA8ABAB,ABABA8AFA0BF80FF
2520 DATA F80CF60BF515D5D5,D5D515F505FD01FF,FEFEFE00EFEFEF,00000044281010 1,FFFFF
FFF,0182020402020 18,0000003070D1F0307,0F1F1F0F07040C
2530 DATA 000080C0E0C080C,E0F2F4E4D448C,FEFEFE00EFEFEF,FFFFFFFF,,,000001030703 01
03,074F2F272B1203,0000C0E0B0F8C0E,F0F8F8F0E0203
2540 DATA FEFEFE00EFEFEF,FFFFFFFF,,,000000040C1D373F,3F1D08,00000060F2FEF8F8,FAF
EF0680478
2550 RESTORE 2460
2560 FOR I=48 TO 57 :: READ A$ :: CALL CHAR(I,A$):: NEXT I
2570 FOR I=64 TO 90 :: READ A$ :: CALL CHAR(I,A$):: NEXT I
2580 FOR I=112 TO 143 :: READ A$ :: CALL CHAR(I,A$):: NEXT I
2590 FOR I=96 TO 101 :: CALL CHAR(I,"FFC399F9F3E7FFE7"):: NEXT I
2600 CALL CHAR(35,"3C428189919152 3C183418BCBEFE2436"):: CALL CHAR(38,"080C0EFFFF
0E0C08")
2610 CALL CHAR(40,"187EFE7E181818183018 98F87C0E070340E0703C1E1F0F07")
2620 !@P+
2630 SUBEND
```

TI 99

ASC II - DEF

ASC II - DEF
Definition von Charakteren

Schon lange habe ich mich darüber geärgert, daß beim TI 99/4 A die Belegung der Grafik nach sehr mühsamer Hexadezimaler Belegung erzeugt werden kann. Ich habe mir überlegt, ob es nicht eine andere Möglichkeit der Grafikbelegung gibt.

Diese Möglichkeit habe ich gefunden, indem ich die Grafikgenerierung durch den Rechner selbst machen lasse. Dazu brauche ich nur mit dem Cursor in einer vorgegebenen Maske die einzelnen Kästchen schwarz zu färben, und der TI generiert sich seine Grafik selbst. Danach kann ich die so definierte Grafik auf eine Datei abspeichern und in einem Programm aufrufen. Dies hat noch den Vorteil, daß ich dadurch eine Menge Speicherplatz einspare, wodurch ich im Speicher noch längere Programme unterbringen kann.

Ebenso kann ich den graphisch belegten ASC II-Satz auf dem Drucker ausgeben und sogar eine Hardcopy mit dem vom Benutzer definierten Zeichensatz erstellen.

Aus diesen vorgenannten Betrachtungen entstand nun das nachfolgende sehr leistungsfähige Programmpaket für den TI 99/4 A.

Bei diesem Programmpaket wurden die Fähigkeiten des TI voll ausgenutzt und das mit Extended BASIC. Denn bei einer geschickten Programmierung kann auch in EX-BASIC eine hohe Rechengeschwindigkeit erreicht werden.

So werden Anwender eines Diskettenlaufwerkes eine Hardcopy mit vorher definierter ASCII-Datei in ca. 2 1/2 Minuten erstellen können.

Das Programmpaket ist so gestaltet, daß es sowohl mit Diskettenlaufwerk als auch mit Cassettenrecorder gefahren werden kann. Jedoch ist die Verarbeitung mittels Cassette wesentlich langsamer als mit Diskette.

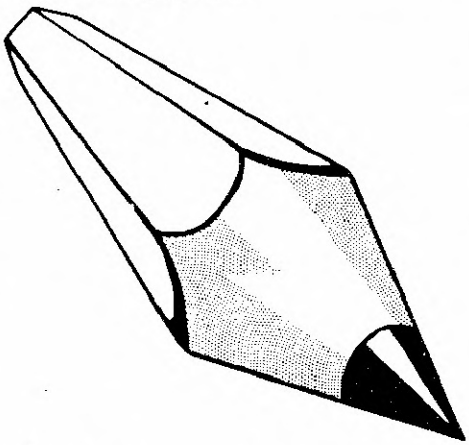
Das Paket wurde extra in mehrere Programme unterteilt, damit keine Speichererweiterung erforderlich ist. Das Programmpaket kann deshalb auch mit der Grundkonfiguration (Konsole, Fernseher und Recorder) gefahren werden.

Selbst ohne externen Speicher ist das Programm ASCIIDEF durch seine vielseitigen Grafikmöglichkeiten noch interessant.

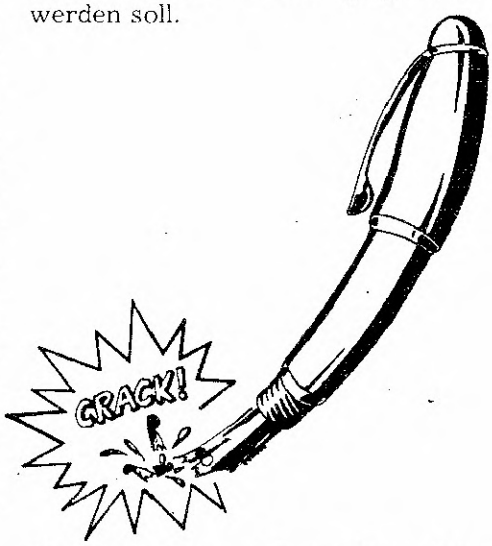
Als Drucker wurde ein Seikosha GP 700 A verwendet. Es kann jedoch auch jeder andere Drucker mit 8 DOT Matrix verwendet werden. Dazu sind evtl. nur die Druckeroptionen zu ändern, die aus diesem Grunde als Stringvariable am Anfang der Pro-

gramme ASCIIPRINT und COPYPRINT stehen.
Auf eine Farbsteuerung des Druckers wurde bewußt verzichtet, da dies evtl. Schwierigkeiten bei der Umsetzung auf andere Drucker geben könnte.
Das Programmpaket besteht aus folgenden Programmen:
ASCIIDEF (Definition von Charakteren)
ASCIIPAT (Unterprogramm zur autom. Definition)
ASCIISET (Umbelegung des Grafiksatzes mit Datei)
ASCIIUMW (Umwandeln für die Druckerausgabe)
ASCIIPRINT (Ausdruck des graphischen ASCII Satzes)
COPY (Unterprogramm zur Speicherung d. Hardcopy)
COPYPRINT (Umwandeln u. Ausdruck der Hardcopy)

Programm ASCIIDEF
Das Programm ASCIIDEF erlaubt es dem Programmierer, einen eigenen Zeichensatz im Bildschirmdialog zu erstellen sowie einen bereits erstellten Zeichensatz zu verändern.
Die definierten Zeichen können gespiegelt werden sowie nach rechts oder links gedreht werden. Es können Zeichen in einer 8x8-Matrix (1 Charakter) als auch in einer 16x16-Matrix (4 Charaktere für Sprites im MAGIFY(3)-Modus) definiert und editiert werden. Auch kann man einzelne Teile einer 16x16-Matrix verändern.
Die so erstellte Datei kann dann mittels des Unterprogramms ASCIISET zur Umbelegung des Zeichensatzes verwendet werden.
Laden Sie das Programm ein und starten Sie es mit RUN. Das Programm führt Sie in Dialog weiter. Zuerst erscheint auf dem Bildschirm das Wort INITIALISIEREN . . . , dabei wird die Datei mit dem Standardzeichensatz des TI 99/4 A belegt. Danach erscheint GRAPHIK BELEGEN . . . , dabei wird die im Programm benötigte Grafik generiert. Jetzt erscheint die Frage NEUES CODEFELD ERSTELLEN? Wird die Frage mit ‚N' beantwortet, so kann jetzt ein bereits vorhandenes Codefeld von Diskette oder Cassette geladen werden. Dazu wird nach Aufforderung der Dateiname (z. B.: DSK1, FELD oder CS1) eingegeben.
Ist die Antwort jedoch ‚J', so wird als Codefeld der Standard-Zeichensatz des TI vorausgesetzt. Jetzt wird gefragt ‚16x16-PUNKTMATRIX'. Antworten Sie mit ‚J', so erscheint diese Matrix auf dem Bildschirm. Wenn Sie mit ‚N' antworten, so wird eine 8x8-Matrix ausgegeben.



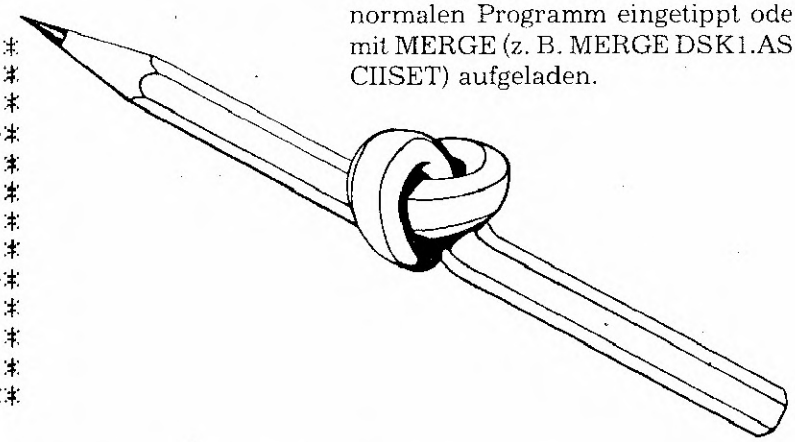
Als nächstes werden Sie gefragt, ob Sie einen Charakter ändern wollen. Wenn die Antwort ‚J' lautet, so müssen Sie die Nummer des zu ändernden Zeichens angeben. Bei einer 16x16-Matrix kann diese Nummer nur dem ersten oder fünften Zeichen eines der 14 Charaktersätze des Rechners entsprechen, da sonst bei einem späteren Spriteaufruf in anderen Programmen das Bild verdreht wiedergegeben würde. Dies wird jedoch vom Programm kontrolliert und gegebenenfalls mit einer Fehlermeldung quittiert, die dazu führt, daß man sich nochmals dazu entscheidet, ob der Charakter geändert werden soll oder nicht. Der Charakter wird jetzt auf der großen Matrix generiert und das Programm springt in den EDIT-Modus. Wenn keine Änderung eines Charakters erwünscht ist, so springt das Programm bei einer weißen Matrix in den EDIT-Modus.
Im Edit-Modus kann der Charakter definiert werden:
Taste ‚1' färbt das Kästchen schwarz.
Taste ‚0' färbt das Kästchen weiß.
Taste ‚D' bewegt den Cursor nach rechts.
Taste ‚S' bewegt den Cursor nach links.
Taste ‚E' bewegt den Cursor nach oben.
Taste ‚X' bewegt den Cursor nach unten.
Um den Editiermodus zu beenden, drücken Sie die Tasten ‚FCTN' und ‚BACK' gleichzeitig und Sie erhalten das Menue:
‚1' – ZEICHEN GENERIEREN
‚2' – ZEICHEN SPIEGELN
‚3' – ZEICHENDREHUNG RECHTS
‚4' – ZEICHENDREHUNG LINKS
ZEICHEN UND SPRITEDEFINITION
Bei Auswahl ‚2' wird das generierte Zeichen gespiegelt.
ZEICHEN UND SPRITEDEFINITION
Die Auswahl ‚3' hat zur Folge, daß das Zeichen im Uhrzeigersinn um 90 Grad gedreht wird.
ZEICHEN UND SPRITEDEFINITION
Auswahl ‚4' dreht das Zeichen um 90 Grad gegen den Uhrzeigersinn.
ZEICHEN UND SPRITEDEFINITION
Bei Auswahl ‚1' wird der Hexadezimale Code des definierten Zeichens erzeugt und ausgegeben. Um fortzufahren, müssen Sie die Tasten ‚FCTN' und PROC/D gleichzeitig drücken.
Es erscheint nun die Frage: CHARAKTER ABSPEICHERN. Bei Antwort ‚J' werden Sie aufgefordert, die Kennzahl des Zeichens einzugeben, unter welcher dieses gespeichert werden soll.



Nach sämtlichen 4 Auswahlmöglichkeiten werden Sie jetzt gefragt: WEITERE OPERATIONEN. Diese Frage bezieht sich auf das z. Z. auf dem Bildschirm sichtbare Zeichen.
Wenn Sie die Frage mit ‚J' beantworten, kommen Sie wieder in den EDIT-Modus.

Wird die Frage mit ‚N' beantwortet, so werden Sie nach weiteren Definitionen gefragt. Antworten Sie mit ‚J', um zu der Frage 16x16 PUNKTMATRIX zu kommen. Wenn Sie jedoch mit ‚N' antworten, so haben Sie jetzt die Auswahl, die soeben definierte Datei zu speichern. Auch ohne Speicherung und Weiterverarbeitung der Datei ist das Programm ASCIIDEF ein sehr hilfreiches Werkzeug, welches die mühsame Arbeit der Charakterdefinition in Hexadezimalcode abnimmt.

Wenn Sie bereits ein Programm haben, worin Charaktere belegt sind, so können Sie sich das Codefeld dieses Programms mittels des Unterprogramms ASCIIPAT abspeichern und evtl. auch editieren, ausdrucken oder Hardcopys dieses Programms ziehen. Tippen Sie das Unterprogramm ASCIIPAT zusätzlich in Ihr Programm ein oder laden dieses mit MERGE (nur mit Diskettenlaufwerk) auf.
Um das Codefeld zu erhalten, rufen Sie das Unterprogramm nach der letzten CHAR-Belegung mit CALL ASCIIPAT(„NAME") auf, wobei „NAME" für die entsprechende Datei (z. B. DSK1.FELD" oder „CS1") steht.
Die so erstellte Datei hat genau die gleiche Form wie das Codefeld, welches mittels „ASCIIDEF" erstellt wurde und ist somit vollends kompartibel zu dieser Datei.
Um in einem Programm den Zeichensatz mittels eines zuvor erstellten Codefeldes umzubelegen, wird das Unterprogramm ASCIISET verwendet. Das Programm wird zusätzlich zum normalen Programm eingetippt oder mit MERGE (z. B. MERGE DSK1.ASCIISET) aufgeladen.

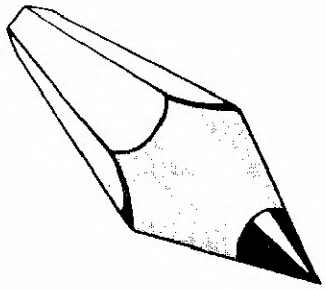


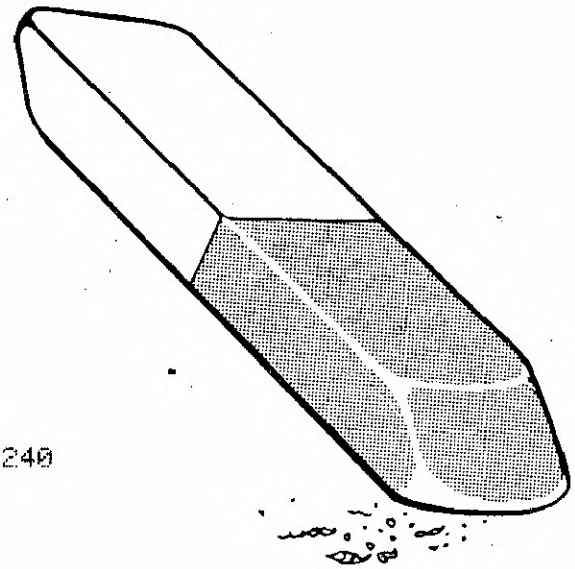
```
10 REM ******************************
12 REM *        A S C I I D E F     *
14 REM * DEFINITION VON CHARAKTEREN  *
16 REM *----------------------------*
18 REM *   T I  9 9 / 4  A          *
20 REM * + TI Extended BASIC         *
22 REM * + externer Speicher         *
24 REM *   (DISC Laufwerk o.Recorder *
26 REM *----------------------------*
28 REM * (c) 1984 by Volker Becker   *
30 REM *     Steinbacher Strasse 10  *
32 REM *     6370 Oberursel 6        *
34 REM ******************************
36 REM
38 REM
100 DISPLAY AT(1,1)ERASE ALL:"ZEICHEN UND SPRITEDEFINITION-----------------------
------"
110 DISPLAY AT(4,1):"INITIALISIEREN....."
120 CALL CHARSET
130 DIM ASCII$(112),FELD(8),FELD$(16)
140 FOR I=32 TO 127 :: CALL CHARPAT(I,HEX$):: ASCII$(I-32)=HEX$ :: NEXT I
150 FOR I=128 TO 143 :: ASCII$(I-32)=RPT$("0",16):: NEXT I
155 FOR I=1 TO 8 :: FELD(I)=0 :: NEXT I
160 DISPLAY AT(4,20):"BEENDET"
170 DISPLAY AT(5,1):"GRAFIK BELEGEN....."
180 CALL CHAR(100,RPT$("0",16),101,RPT$("F",16),102,"FF818181818181FF"):: CALL C
OLOR(9,2,16)
190 DISPLAY AT(5,20):"BEENDET"
200 DISPLAY AT(8,1):"NEUES CODEFELD ERSTELLEN ?  (J/N)" :: ACCEPT AT(9,7)SIZE(1)
VALIDATE("JN")BEEP:SKP$ :: IF SKP$="J" THEN 250
210 DISPLAY AT(10,1):"FELDNAME ? (GERAET.NAME)" :: ACCEPT AT(11,1)BEEP:DATEI$
220 OPEN #1:DATEI$,INPUT ,SEQUENTIAL,INTERNAL,FIXED 80
230 FOR I=32 TO 143 :: INPUT #1:GRAPH$,ASCII$(I-32):: NEXT I
240 CLOSE #1
245 DISPLAY AT(1,1):"ZEICHEN UND SPRITEDEFINITION-----------------------------"
250 CALL DELSPRITE(#1):: CALL CHAR(96,RPT$("0",64)):: CALL HCHAR(3,1,32,704):: C
ALL SCREEN(4)
260 DISPLAY AT(24,1)BEEP:"16 X 16 PUNKTMATRIX ? (J/N)"
270 ACCEPT AT(24,28)VALIDATE("JN"):SKP$
280 IF SKP$="J" THEN CALL M16 ELSE CALL M8
290 IF SKP$="J" THEN MTX=1 ELSE MTX=0
320 DISPLAY AT(24,1):"CHARAKTER AENDERN ? (J/N)"
330 ACCEPT AT(24,28)SIZE(1)VALIDATE("JN")BEEP:SKP$
340 IF SKP$="J" THEN 20000
350 IF MTX=1 THEN CALL EDIT(20,20)ELSE CALL EDIT(12,12)
360 CALL DELSPRITE(#1)
```

```
370 DISPLAY AT(21,1):"'1' - ZEICHEN GENERIEREN    '2' - ZEICHEN SPIEGELN      '3
' - ZEICHENDREHUNG RECHTS '4' - ZEICHENDREHUNG LINKS"
380 ACCEPT AT(19,26)VALIDATE("1234")SIZE(1)BEEP:SKIP
390 ON SKIP GOSUB 2000,3000,4000,4000
400 CALL HCHAR(21,1,32,128):: CALL HCHAR(19,28,32):: DISPLAY AT(24,1):"WEITERE O
PERATIONEN ? (J/N)" :: ACCEPT AT(24,28)SIZE(1)VALIDATE("JN")BEEP:SKP$
410 IF SKP$="J" THEN 350
420 DISPLAY AT(24,1):"WEITERE DEFINITIONEN (J/N)" :: ACCEPT AT(24,28)SIZE(1)VALI
DATE("JN")BEEP:SKP$ :: IF SKP$="J" THEN 250
430 CALL DELSPRITE(#1):: CALL HCHAR(3,1,32,704):: DISPLAY AT(4,1):"CODEFELD ABSP
EICHERN? (J/N)" :: ACCEPT AT(4,28)VALIDATE("JN")SIZE(1)BEEP:SKP$
440 IF SKP$="N" THEN 1000
450 DISPLAY AT(6,1):"FELDNAME ? (GERAET.NAME)"
460 ACCEPT AT(7,1)BEEP:DATEI$
470 OPEN #1:DATEI$,OUTPUT,SEQUENTIAL,INTERNAL,FIXED 80
480 FOR I=32 TO 143 :: PRINT #1:"",ASCII$(I-32):: NEXT I
490 CLOSE #1
1000 CALL CLEAR :: STOP :: END
2000 ! GENERIEREN DER ZEICHEN
2010 UHEX$="0123456789ABCDEF"
2020 HEX$=""
2025 CALL HCHAR(21,1,32,128):: CALL HCHAR(19,28,32)
2030 IF MTX=0 THEN GOTO 2100
2040 BS=5 :: BZE=20 :: GOSUB 5000
2050 BS=13 :: GOSUB 5000
2060 CALL CHAR(96,HEX$):: CALL MAGNIFY(4):: CALL SPRITE(#1,96,2,41,185)
2062 FOR I=0 TO 3 :: CALL CHARPAT(96+I,HEX$):: DISPLAY AT(21+I,3):HEX$ :: NEXT I
2070 GOTO 2200
2100 BS=5 :: BZE=12
2110 GOSUB 5000
2120 CALL CHAR(96,HEX$):: CALL MAGNIFY(2):: CALL SPRITE(#1,96,2,33,121)
2130 DISPLAY AT(21,3):HEX$
2200 DISPLAY AT(21,22):"WEITER" :: DISPLAY AT(22,22):" MIT" :: DISPLAY AT(23,22)
:"PROC'D"
2202 CALL KEY(5,T,S):: IF T<>12 THEN 2202
2208 CALL HCHAR(21,1,32,128):: CALL HCHAR(19,26,32):: DISPLAY AT(24,1):"CHARAKTE
R ABSPEICHERN (J/N)" :: ACCEPT AT(24,28)VALIDATE("JN")SIZE(1)BEEP:SKP$
2210 IF SKP$="J" THEN 2220 ELSE RETURN
2220 DISPLAY AT(24,1):"UNTER WELCHEM ZEICHEN ?" :: ACCEPT AT(24,25)SIZE(3)VALIDA
TE(DIGIT)BEEP:CHARNR
2230 IF CHARNR<32 OR CHARNR>143 THEN 2220
2240 IF MTX=1 THEN BACK=1 :: GOSUB 24000
2245 IF MTX=2 THEN MTX=1 :: GOTO 2200
2250 IF MTX=1 THEN 2270
2260 CALL CHARPAT(96,ASCII$(CHARNR-32)):: RETURN
2270 FOR I=0 TO 3
2280 CALL CHARPAT(96+I,ASCII$(CHARNR-32+I))
2290 NEXT I
2300 RETURN
3000 ! ZEICHEN SPIEGELN
3010 PS=5
3020 IF MTX=1 THEN PE=20 :: L=15 ELSE PE=12 :: L=7
3030 FOR I=5 TO L+5
3040 CALL GCHAR(I,PS,Z1):: CALL GCHAR(I,PE,Z2):: CALL HCHAR(I,PS,Z2):: CALL HCHA
R(I,PE,Z1)
3050 NEXT I
3060 PS=PS+1 :: PE=PE-1
3070 IF PE<PS THEN RETURN
3080 GOTO 3030
4000 ! DREHEN DES ZEICHENS
4010 FOR I=1 TO 16 :: FELD$(I)="" :: NEXT I
4020 IF MTX=1 THEN PS=5 :: PE=20 ELSE PS=5 :: PE=12
4030 IF SKIP=4 THEN 4200
```

```
4040 FOR I=PS TO PE
4050 FOR K=PE TO PS STEP -1
4060 CALL GCHAR(K,I,Z1)
4070 FELD$(I-PS+1)=FELD$(I-PS+1)&CHR$(Z1)
4080 NEXT K
4090 NEXT I
4100 GOTO 4500
4200 J=0
4205 FOR I=PE TO PS STEP -1
4207 J=J+1
4210 FOR K=PS TO PE
4220 CALL GCHAR(K,I,Z1)
4230 FELD$(J)=FELD$(J)&CHR$(Z1)
4240 NEXT K
4250 NEXT I
4500 FOR I=PS TO PE
4510 FOR K=1 TO PE-PS+1
4520 CALL HCHAR(I,K+4,ASC(SEG$(FELD$(I-PS+1),K,1)))
4530 NEXT K
4540 NEXT I
4550 RETURN
5000 ! BINAER - HEXADEZIMAL - WANDLER
5010 FOR K=5 TO BZE
5020 FOR I=BS TO BS+7
5030 CALL GCHAR(K,I,F):: FELD(I-BS+1)=F-100
5040 NEXT I
5050 LOW=FELD(5)*8+FELD(6)*4+FELD(7)*2+FELD(8)+1
5060 HIGH=FELD(1)*8+FELD(2)*4+FELD(3)*2+FELD(4)+1
5070 HEX$=HEX$&SEG$(UHEX$,HIGH,1)&SEG$(UHEX$,LOW,1)
5080 NEXT K
5090 RETURN
20000 DISPLAY AT(23,1):" " :: DISPLAY AT(24,1):"WELCHEN CHARAKTER ?" :: ACCEPT A
T(24,21)SIZE(3)VALIDATE(DIGIT)BEEP:CHARNR
20005 BACK=0
20010 IF CHARNR<32 OR CHARNR>143 THEN 21000
20020 FOR I=1 TO 16 :: FELD$(I)="" :: NEXT I
20030 IF MTX=1 THEN 22000
20032 ASCI$=ASCII$(CHARNR-32)
20040 CALL ASCII(FELD$(),ASCI$)
20050 CALL CHARPRT(5,5,FELD$())
20060 CALL CHAR(96,ASCII$(CHARNR-32)):: GOTO 350
21000 DISPLAY AT(23,1):"NICHT MOEGLICH !!!" :: CALL SOUND(250,110,0)
21010 IF BACK=1 THEN MTX=2 :: RETURN ELSE 320
22000 DATA 5,5,13,5,5,13,13,13
22010 RESTORE 22000
22020 GOSUB 24000
22030 FOR I=0 TO 3
22035 FOR J=1 TO 16 :: FELD$(J)="" :: NEXT J
22040 ASCI$=ASCII$(CHARNR-32+I)
22050 CALL ASCII(FELD$(),ASCI$)
22060 READ P1,P2 :: CALL CHARPRT(P1,P2,FELD$())
22070 CALL CHAR(96+I,ASCII$(CHARNR-32+I))
22080 NEXT I
22090 GOTO 350
24000 ! KONTROLLE DER MAGNIFY(3) POSITIONEN
24010 FOR I=32 TO 140 STEP 4
24020 IF CHARNR=I THEN RETURN
24030 NEXT I
24040 GOTO 21000
25000 SUB M8 ! 8 X 8 MATRIX ENTSPR. 1 CHARAKTER
25010 FOR I=5 TO 12 :: CALL HCHAR(I,5,100,8):: NEXT I
25020 FOR I=1 TO 8 :: CALL HCHAR(4,4+I,I+48):: CALL VCHAR(4+I,4,I+48):: NEXT I
25030 CALL HCHAR(3,16,96):: FOR I=5 TO 6 :: CALL HCHAR(I,16,100,2):: NEXT I
```

```
25040 SUBEND
25050 SUB M16 ! 16 X 16 MATRIX ENTSPR. 4 CHARAKTEREN
25060 FOR I=1 TO 16 :: CALL HCHAR(4+I,5,100,16):: NEXT I
25070 CALL HCHAR(3,5,48,9):: CALL HCHAR(3,14,49,7)
25080 FOR I=1 TO 9 :: CALL HCHAR(4,4+I,I+48):: NEXT I
25090 FOR I=0 TO 6 :: CALL HCHAR(4,14+I,I+48):: NEXT I
25100 CALL VCHAR(5,3,48,9):: CALL VCHAR(14,3,49,7)
25110 FOR I=1 TO 9 :: CALL VCHAR(I+4,4,I+48):: NEXT I
25120 FOR I=0 TO 6 :: CALL VCHAR(I+14,4,I+48):: NEXT I
25130 CALL HCHAR(3,25,96):: CALL HCHAR(4,25,97):: CALL HCHAR(3,26,98):: CALL HCH
AR(4,26,99)
25140 FOR I=6 TO 9 :: CALL HCHAR(I,24,100,4):: NEXT I
25150 SUBEND
26000 SUB CHARPRT(Z,S,FELD$())
26010 FOR I=Z TO Z+7
26020 FOR K=S TO S+7
26030 CALL HCHAR(I,K,VAL(SEG$(FELD$(I-Z+1),K-S+1,1))+100)
26040 NEXT K
26050 NEXT I
26060 SUBEND
27000 SUB EDIT(XMAX,YMAX)! DEFINIEREN UND EDITIEREN DES CHARAKTERS
27002 CALL DELSPRITE(#1):: CALL MAGNIFY(1):: CALL SPRITE(#1,102,7,33,33)
27005 DISPLAY AT(23,1):"DRUECKEN SIE FCTN 'BACK' UM DIE DEFINITION ZU BEENDEN"
27010 X,Y=5
27020 CALL KEY(5,T,S):: IF S=0 THEN 27020
27040 IF T=48 OR T=49 THEN 27200
27050 IF T=88 THEN Y=Y+1
27060 IF T=83 THEN X=X-1
27070 IF T=68 THEN X=X+1
27080 IF T=69 THEN Y=Y-1
27090 IF T=15 THEN SUBEXIT
27100 IF X>XMAX THEN X=XMAX
27110 IF Y>YMAX THEN Y=YMAX
27120 IF X<5 THEN X=5
27130 IF Y<5 THEN Y=5
27140 CALL LOCATE(#1,(Y-1)*8+1,(X-1)*8+1)
27150 GOTO 27020
27200 CALL HCHAR(Y,X,T+52)
27210 X=X+1 :: IF X>XMAX THEN X=5 :: Y=Y+1
27220 IF Y>YMAX THEN Y=5
27230 GOTO 27140
27240 SUBEND
31999 SUB ASCII(FELD$(),HEX$)
32050 M=1
32060 C$=""
32070 H$=SEG$(HEX$,1,2)
32080 E=0
32090 FOR L=LEN(H$)TO 1 STEP -1
32100 B$=SEG$(H$,L,1)
32110 FOR J=0 TO 5
32120 IF ASC(B$)=65+J THEN A=10+J :: GOTO 32150
32130 NEXT J
32140 A=VAL(B$)
32150 D=D+A*16^E
32160 E=E+1
32170 NEXT L
32180 IF D=0 THEN FELD$(M)="00000000" :: GOTO 32240
32190 D=D/2
32200 IF D=INT(D)THEN C$="0"&C$ :: GOTO 32220
32210 C$="1"&C$ :: D=INT(D)
32220 IF INT(D)>0 THEN 32190
32230 FELD$(M)=C$ :: C$=""
```





```
32240 M=M+1
32250 HEX$=SEG$(HEX$,3,LEN(HEX$)-2)
32260 IF HEX$<>"" THEN 32070
32270 D=0
32280 FOR R=1 TO 8
32290 IF LEN(FELD$(R))<8 THEN FELD$(R)="0"&FELD$(R):: GOTO 32290
32300 NEXT R
32420 SUBEND
```

ASC II - SET Unterprogramm

```
32400 SUB ASCIISET(DATEI$)
32410 !
32420 ! Unterprogramm zur Umbelegung des Graphiksatzes
32430 !
32440 ! fuer II 99/4 A
32450 !
32460 ! (c) 1984 by Volker Becker
32470 !
32480 !
32482 CALL CLEAR
32485 FOR I=32 TO 143
32487 PRINT CHR$(I);" ";
32489 NEXT I
32490 OPEN #88:DATEI$,INPUT ,SEQUENTIAL,INTERNAL,FIXED 80
32500 FOR I=32 TO 143
32510 INPUT #88:GRAPH$,ASCII$
32520 CALL CHAR(I,ASCII$)
32530 NEXT I
32540 CLOSE #88
32550 SUBEND
```

ASC II - PAT Unterprogramm

```
32000 SUB ASCIIPAT(DATEI$)
32005 !
32010 ! copieren eines Codefeldes aus einem bestehenden Programm
32015 !
32020 ! (c) 1984 by Volker Becker
32025 !
32030 ! fuer TI 99/4 A
32035 !
32040 !
32045 OPEN #88:DATEI$,OUTPUT,SEQUENTIAL,INTERNAL,FIXED 80
32050 FOR I=32 TO 143
32055 CALL CHARPAT(I,HEX$)
32060 PRINT #88:"",HEX$
32065 NEXT I
32070 CLOSE #88
32075 SUBEND
```

Achtung: Bei dem Programm ASCIIDEF kann es bei einer falschen Eingabe oder bei einer vollen Diskette am Programmende zu einer Fehlermeldung kommen (I/0 ERROR). Wenn dies geschieht, so kann man die im Rechner befindliche Datei retten, da sonst die ganze Definition umsonst gewesen wäre.
Dazu gibt man als Befehl, nachdem die Fehlermeldung gekommen ist, die nachfolgende Zeile ein:
OPEN 7: „Gerät. Name", OUTPUT, SEQUENTIAL, INTERNAL, FIXED 80 : : FOR I=32 TO 143 : : PRINT 7: „,ASCII$ (I-32) : : NEXT I
Danach ENTER drücken. Der Rechner rettet jetzt die Datei auf den entsprechenden externen Speicher. (Siehe Programm ASCIIDEF Zeilen 470 und 480).
Dies geht jedoch nur, wenn noch kein anderer Befehl nach der Fehlermeldung gegeben wurde.

C 64

Listschutz

In der Juni-Ausgabe haben wir bereits eine List-Schutz-Routine vorgestellt. Leider hat sich erst später herausgestellt, daß einige Basic-Befehle nach dem Schutz nicht mehr ausgeführt werden. Dies kommt durch die Arbeitsweise des Basic-Interpreters und kann nicht verhindert werden.
Die nun hier vorgestellte Schutz-Routine ist etwas arbeitsaufwendiger, dafür jedoch in fast allen Zeilen zu verwenden. Der Vorteil ist, daß sich auch einzelne Zeilen schützen lassen. Beim Listen von geschützten Zeilen erscheint nur die Zeilen-Nummer.
Und so wird's gemacht:
Vor jede zu schützende Zeile wird 5 mal die eckige Klammer gesetzt (hinter Zeilennummer und vor Befehl).
Beispiel:
30[[[[[PRINT „DIESE ZEILE IST GESCHUETZT"
Die zu schützenden Basic-Programme sollten mit einer Zeilennummer >5 beginnen, da in den ersten Zeilen die Schutz-Routine eingetragen wird.
Schutzroutine:

```
1 N=2049
2 IF PEEK(N+4)=ASC("[") THEN POKE N+4,0
3 N= PEEK(N)+256 · PEEK(N+1)
4 IF N>0 THEN 2
5 END
```

Durch den Befehl „RUN" werden nun die gekennzeichneten Zeilen geschützt. Die Befehle GET, INPUT und DEFFN dürfen nicht geschützt werden.

2 × Spectrum

Ein Scroll wie beim ZX-81 kann man in Programmen durch >RANDOMIZE USR 3280< ersetzen.

Die fre (0) Funktion fehlt beim ZX-Spectrum. Man kann die noch freien Bytes folgendermaßen erfahren: Print 65536 – USR 7962

24-Zeilen-COPY für den Sinclair ZX Spectrum

Wie man die Bildschirmzeilen 22 und 23 nutzen kann, stand bereits in verschiedenen Fachzeitschriften und soll daher hier nur kurz erwähnt werden:

Zeile 22: PRINT O; AT O,...
Zeile 23: PRINT O; AT O,...

Schwieriger wird es dagegen, wenn man den Bildschirm einschließlich der Zeilen 22 und 23 auf dem Drucker ausgeben will. Die COPY-Routine des Spectrum-ROM verarbeitet nämlich nur die 22 Zeilen des oberen Bildschirmteils.
Abhilfe schafft hier ein kleines Programm, das die beiden Zeilen des unteren Bildschirmteils nacheinander in den Printerbuffer kopiert und dann mit LPRINT auf dem Drucker ausgibt.
Das eigentliche Programm zum Kopieren und Ausdrucken der Zeilen steht in den Programmzeilen 110 bis 140. Es sieht auf den ersten Blick vielleicht etwas kompliziert aus, aber es hat den Vorteil, daß es ohne Veränderung mit beliebigen Zeilennummern versehen werden kann. Außerdem spielt der unterschiedliche Aufbau des Display-Files und des Printerbuffers eine Rolle.
Um in einem Programm die 24-COPY durchzuführen, setzt man einfach die Zeilen 110 bis 140 hinter den eigentlichen COPY-Befehl.

```
        Zeile 0
        Zeile 1
        Zeile 2
        Zeile 3
        Zeile 4
        Zeile 5
        Zeile 6
        Zeile 7
        Zeile 8
        Zeile 9
        Zeile 10
        Zeile 11
        Zeile 12
        Zeile 13
        Zeile 14
        Zeile 15
        Zeile 16
        Zeile 17
        Zeile 18
        Zeile 19
        Zeile 20
        Zeile 21
Das ist die Zeile 22
     und das ist die Zeile 23
```

```
  1 REM ***********************
  5 REM *                     *
 10 REM *    24-Zeilen-COPY   *
 20 REM *                     *
 30 REM *                     *
 40 REM * ©1984 JOACHIM MILTZ *
 50 REM *                     *
 60 REM *      Hauptstr. 96   *
 70 REM *      5401  Wolken   *
 80 REM *                     *
 90 REM ***********************
 95 REM
100 CLS : GO TO 300
105 COPY
110 FOR z=20672 TO 20704 STEP 3
2
115 LET z1=z
120 FOR i=0 TO 7
125 FOR j=0 TO 31
130 POKE 23296+j+i*32,PEEK (z1+
j)
135 NEXT j: LET z1=z1+256
140 NEXT i: LPRINT : NEXT z
190 PAUSE 100: STOP
300 REM   DEMONSTRATION
310 FOR i=0 TO 21
330 PRINT AT i,6;"Zeile ";i
340 NEXT i
350 PLOT 0,0: DRAW 255,0: DRAW
0,175. DRAW -255,0. DRAW 0,-175
360 PRINT #0;AT 0,0;"Das ist di
e Zeile 22"
365 PRINT #0;AT 1,6;"und das is
t die Zeile 23"
370 GO TO 105
900 SAVE "COPY 24" LINE 100
910 VERIFY "COPY 24"
```

TI-99

Richtigstellung der in Computronic Heft 6 auf Seite 5 abgedruckten >Tips & Tricks< für den TI-99/4A.

Call Files

Beschreibung:
Das FILES-Unterprogramm verwendet man zur Änderung der Anzahl der externen Dateien, die gleichzeitig geöffnet sein können. Die Anzahl muß eine einstellige ganze Zahl (von 1 bis 9) sein. Der Befehl NEW muß im Direktbetrieb unmittelbar dem CALL FILES-Befehl folgen. Wird der CALL FILES-Befehl nicht eingegeben, so ist der Standardwert der Anzahl der möglichen offenen Dateien 3, sofern ein Peripheriegerät angeschlossen ist.
Das FILES-Unterprogramm ist vor allem im Zusammenhang mit dem Diskettenbetrieb interessant. Pro geöffneter Datei wird der für Programme verfügbare Speicherplatz im RAM-Bereich des Computers um 518 Bytes reduziert. (Bei zugeschaltetem Disk-Controller kann also der verfügbare Speicherplatz mit CALL FILES(1) oder CALL FILES(2) erhöht werden!)

Achtung!
Im TI-BASIC ist der CALL FILES-Befehl nur im Direktbetrieb (zusammen mit NEW) möglich. Die Anwendung des FILES-Unterprogramms in einem Programm kann unvorhersehbare Folgen haben – einschließlich Verlust eines Programmes und/oder Verlust von Daten auf der Diskette! CALL FILES (Anzahl) ist in den Peripheriegeräten implementiert. Es existiert nicht bei bloßem Anschluß eines Kassettenrecorders.

CALL FILES – Das FILES-Unterprogramm verwendet man zur Änderung der Anzahl der Dateien (max. 9), die gleichzeitig offen sein können. Der Standardwert der zulässigen offenen Dateien in TI BASIC liegt bei drei. Der für Programme verfügbare Speicherplatz im RAM wird um 534 Byte reduziert, plus 518 weitere Bytes für jede zulässige offene Datei.
Der CALL FILES Befehl hat folgende allgemeine Form:

CALL FILES (Anzahl)
NEW

Die Anzahl muß eine einstellige Zahl von 1 bis 9 sein. Sie gibt die Menge der Dateien an, die gleichzeitig offen sein können. Der Befehl NEW muß unmittelbar nach dem CALL FILES Befehl folgen. Wird CALL FILES nicht durchgeführt, liegt der Standardwert der möglichen offenen Dateien bei drei.

Vorsicht
CALL FILES kann nur als Befehl in TI BASIC verwendet werden. Es ist nicht zulässig, CALL FILES als Statement (Anweisung) in einem TI BASIC Programm einzugeben. Die Anwendung von CALL FILES in einem Programm ohne das unmittelbar folgende NEW kann unvorhergesehene Folgen haben, einschließlich Verlust eines Programms und/oder Verlust von Daten auf einer Diskette.

Erfreulich

Hiermit möchte ich Ihnen mitteilen, daß ich die Nachlieferung der Apple Disketten # 4 und # 5 erhalten habe. Weiterhin möchte ich mich bei Ihnen für den schnellen Service vielmals bedanken! Es ist heutzutage leider nicht mehr üblich auf Reklamationen so schnell zu reagieren und Abhilfe zu leisten. Deshalb VIELEN DANK!!! Holger Zeißler, Alsbach

Ärgerlich

Zu den Programmen „Schneevogel" Heft 6 und „Chopper-Flight" Heft 7, für Atari

Zu meinem großen Erstaunen habe ich festgestellt, daß in dem Listing „Schneevogel" für Atari-Computer in „Computronic" Nr. 6, ein Richard Käfer als Copyright-Inhaber angegeben ist. Dies entspricht jedoch nicht der Wahrheit!
Dieses Spiel ist schon im Januar '84 in einem amerikanischen Magazin erschienen. Denzufolge liegen die Urheberrechte bei dem dortigen Autor. Ich bitte Sie um Überprüfung dieser Tatsache! Jörg Detlefs, Hamburg

Ich möchte Sie hiermit darauf aufmerksam machen, daß das Listing „Schneevogel" nicht von Richard Käfer, sondern von John Weber, stammt. Das Listing ist vor längerer Zeit in einer amerikanischen Zeitschrift unter dem Namen „Snowbird" erschienen. Lars Kubitz, Berlin

Als Anlage eine Kopie des Originalprogrammes „Chopper-Flight" aus Amerika („Centurion"). Das Programm soll nämlich in Rot auf dem Bildschirm erscheinen. Das Bild ist aber nur schwarz/weiß. Poke 755, 224 ist falsch. Joachim Maiter

Redaktion:
Zunächst bedanken wir uns für die freundlichen Hinweise, wonach die Listings „Schneevogel" und „Chopper-Flight" für Atari, Heft 6+7, offensichtlich einen anderen Copyright-Inhaber haben als der von uns angegebene.
Unsere Vertragsbedingungen sehen vor, daß der Autor uns die Zusicherung gibt, daß er der alleinige Verfasser ist und dementsprechend über die weitere Verwendung eigenständig bestimmen kann.
Diese Zustimmung wurde uns erteilt. Wir können natürlich nicht jede auf dem Markt erscheinende Computer-Zeitschrift auf den Inhalt überprüfen. Insofern dürfen wir uns auf die zunächst gemachten Angaben verlassen. Allerdings wird unser Haus es nicht erlauben, daß mit solchen Praktiken wir und Sie, lieber Leser, getäuscht werden. Wir werden daher eine Prüfung durchführen, die unter Umständen zu einem Schadenersatz führen wird.

Ergänzung

Für die Veröffentlichung meines Mathematik-Programms möchte ich mich ganz herzlich bei Ihnen bedanken. Gleichzeitig bitte ich aber, den folgenden Nachtrag noch in einer Ihrer nächsten Ausgaben zu bringen! Grund dafür sind Bedienungsfehler Ihrerseits bei den Beispielen, die die Leistungsfähigkeit des Programms bei weitem nicht ausschöpfen. Vielleicht liegt es auch an der Knappheit der Dokumentation, die für „Hobbymathematiker" etwas unzureichend ist.

Nachtrag zum Mathematik-Programm – für ZX-Spektrum
1. Bei der Funktion $y = x^3 - 8x + x^2$ (S. 36) fällt auf, daß diese sich nur auf einen kleinen Teil der X-Achse verteilt. Grund dafür ist eine viel zu hohe Vergrößerung, so daß nur ein Teil der X – Y Paare abgebildet wird, und der letzte Pixel bei x = 2.5 gedruckt wird, obwohl der Definitionsbereich bis 15 gehen sollte. Hier fehlt also ein Teil der Funktion. Wird eine zu kleine Vertikaldehnung gewählt, so wird der GaPh nur etwas kleiner abgebildet, wie zum Beispiel die Sinusfunktion auf Seite 36. In allen Fällen stimmen aber die Koordination, angegeben mit A, B, P und Q!!
Das Problem der optimalen Vergrößerung könnte zwar softwaremäßig leicht gelöst werden, jedoch sehr auf Kosten der Geschwindigkeit. Außerdem ergäben sich große Probleme mit Unstetigkeitsstellen. Auf jeden Fall liegt hier kein Programmierfehler vor, was man bei einer neunten Version auch ausschließen sollte.
2. Auch wenn das Programm weitgehend immun gegen Unstetigkeitsstellen ist, so kann ein Auftauchen der Fehlermeldung „Number too big", oder ein Hinweis auf eine mögliche Definitionslücke nicht ausgeschlossen werden, denn manch eine unstetige Funktion kann selbst mit den raffiniertesten Mitteln der höheren Mathematik nicht gelöst werden! Und mein Programm kann nun einmal gewisse Gesetzmäßigkeiten der Mathematik nicht sprengen!
Abschließend möchte ich auf einen Fehler im Inhaltsverzeichnis hinweisen, wo die Lösbarkeit von Differentialgleichungen angepriesen wird. Richtig müßte es an dieser Stelle Differentialrechnung heißen im Sinne der ersten Ableitung.
Soweit die wichtige Ergänzung.
Uwe Haferland

Ein Tip:

Läßt sich mal eine Kassette nicht laden (Meldung: ERROR in DATA oder DATA NOT FOUND, beim TI-99/4A), dann sollte man versuchen die Kassette mit einem Stereogerät und zwar abwechselnd mit dem rechten oder linken Kanal zu laden – dies geht mit Hilfe des „Balance-Reglers" und eines entsprechenden Stereo-Kabels. Franz Sehi, Fürstenfeldbruck

Begeistert

Mit wachsender Begeisterung lese ich Ihr Computronic-Magazin und ich muß sagen „Sie haben eine Menge drauf" mit den abgedruckten Programmen.
Besonders gut fand ich das Programm für den Spectrum. Ich habe nun einige Änderungen am Listing durchgeführt, sodaß es mit dem Kempston-Joystick spielbar ist.
Meine Frage ist jetzt: Gibt es auch für solche Programmiertricks ein Honorar oder nur für ganze Programme?
Ich frage deshalb, weil ich selbst noch nicht in der Lage bin, Programme zu schreiben, sondern nur einige Änderungen oder Verbesserungen machen kann, die der besseren Spielbarkeit dienen. Harald Schüler, 6900 Heidelberg 1

Redaktion:
Diese Frage wird uns häufig gestellt. Natürlich bezahlen wir unseren Lesern auch gute Programmiertricks. Das Honorar dafür richtet sich nach Länge und Qualität des eingesandten Programmiertips.

Service

Meine TI-99-Kassetten habe ich prompt erhalten. Leider kann ich sie nicht lesen. Nach ca. 10 Std. Versuchen mit unterschiedlichen Recordern habe ich aufgegeben. In Abhängigkeit von der eingestellten Lautstärke erhalte ich entweder „NO DATA FOUND" oder „ERROR IN DATA DETECTED". Diese zweite Fehlermeldung kommt entweder gleich am Anfang bei dem etwas längeren Piepser oder erst nach einiger Zeit während des Einlesens. Vielleicht sollte ich noch darauf hinweisen, daß die eingelesenen Daten über den Monitorlautsprecher zu hören waren. Ich hoffe sehr, daß Sie mir helfen können. Dr. A. Ruff, 7400 Tübingen

Redaktion
Zwar sind wir weitgehendst bemüht, bespielte Cassetten bzw. Disketten im einwandfreien Zustand an unsere Kunden weiterzugeben, jedoch kann es auch bei uns vorkommen, daß ein Datenträger nicht einwandfrei versendet wird. Tatsache ist jedoch, daß jeder Datenträger auf seine Funktion überprüft wird. Durch eine Reihe anderer Umstände wie Tonkopfverschmutzung, unterschiedliche Tonkopfeinstellung, magn. Einflüsse etc. kann dieser Ladefehler Ihrer Kassetten jedoch zustande gekommen sein. Unser Angebot: Senden Sie die defekten Kassetten zurück, Sie erhalten umgehend kostenlos neue Exemplare zurück.

Ein Spiel in 378 Bildern

Vor langer Zeit lebte ein Volk, welches MAYA genannt wurde. Dieses Volk soll unermeßliche Schätze besessen haben.

Jetzt endlich hat sich jemand bereiterklärt, nach diesen sagenumwobenen Schätzen zu suchen. Dieser Jemand ist Coconut Joe. Für die abenteuerliche Expedition braucht er jedoch noch einen Partner. Wie wäre es zum Beispiel mit Ihnen? Begleiten Sie Coconut Joe bei seiner Schatzsuche durch den Urwald, und vielleicht haben Sie Glück und kommen in die Wüste zu den Pyramiden der MAYA, wo Sie weitere Schätze finden können.

Das Spiel läuft auf dem TI 99/4 A mit dem Extended BASIC Modul und wird mittels Joystick 1 gesteuert.

Was Sie jetzt lesen, werden Sie wohl kaum glauben wollen, es ist jedoch wahr:

Der Rechner kann in diesem Spiel bis zu 378!! verschiedene Bildschirmszenen darstellen. Ferner verfügt das Spiel über eine hervorragende grafische Darstellung.
Aus Speicherplatzgründen mußte jedoch auf eine Spielanleitung im Programm verzichtet werden, da das Programm den Speicher bis auf 654 Bytes ausnutzt.

Doch nun zum eigentlichen Spiel:
Tippen Sie das Listing ein oder laden Sie das Programm von Cassette. Danach starten Sie mit „RUN". Es dauert jetzt einige Sekunden, bis der Rechner die internen Vorbereitungen zum Spiel getroffen hat. Jetzt erscheint das Titelbild und El Condor Pasa wird gespielt. Um das Spiel zu beginnen, nehmen Sie Joystick 1 in die Hand und drücken den roten Aktionsknopf. Die Musik verstummt und nach wenigen Sekunden erscheint Coconut Joe am rechten Bildschirmrand. Nachfolgend werden nun die einzelnen Spielelemente beschrieben. Das ist die einfachste Art, das Spiel zu erklären.

Coconut Joe

Coconut Joe wird vom Spieler mittels des Joysticks gesteuert: Nach rechts und links. Wenn Sie im Bereich einer Leiter stehen, so bewegen Sie Joe mit dem Joystick nach oben oder unten, die Leiter hinauf oder hinunter.
Wenn Sie den Aktionsknopf drücken, so springt Joe in die Höhe. Gleichzeitiges Bewegen des Joysticks nach rechts oder links läßt Joe in die gewünschte Richtung springen.

Beachten Sie jedoch, daß Coconut Joe nur auf der Urwald- bzw. Wüstenoberfläche springen kann. In den unterirdischen Katakomben kann Joe nicht springen.

Schlange
Die Schlange kommt sowohl im Urwald als auch in der Wüste vor. Wenn Joe diese Schlange überspringt, so bekommen Sie 100 Punkte auf Ihrem Punktekonto gutgeschrieben. Ferner erhöht sich Ihr Überlebensbonus um 300 Punkte.
Berühren Sie jedoch die Schlange, ohne zu springen, so verliert Coconut Joe sein Leben und das Spiel ist zu Ende.

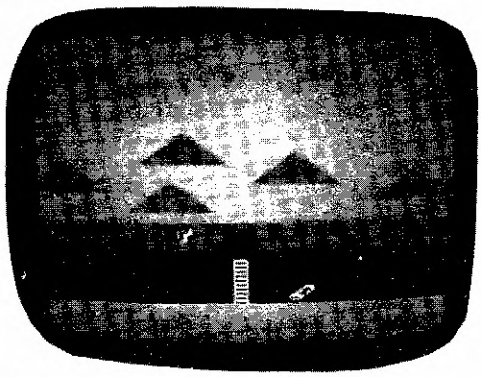
Rollender Baumstamm
Der rollende Baumstamm kommt nur im Urwald vor. Er rollt immer von links nach rechts. Wenn Joe den Baumstamm überspringt, so bekommt er 50 Punkte auf dem Punktekonto gutgeschrieben und 200 Überlebensbonuspunkte.
Wird Joe jedoch vom Baumstamm überrollt, so verliert er sein Leben und das Spiel ist zu Ende.

Urwaldfeuer
Das Urwaldfeuer kommt, wie der Name schon sagt, nur im Urwald vor. Wird es übersprungen, so werden 30 Punkte auf dem Punktekonto gutgeschrieben und der Überlebensbonus erhöht sich um 100 Punkte.
Wird das Feuer jedoch berührt, ohne daß Joe springt, so verliert er sein Leben und das Spiel ist zu Ende.

Kokosnuß
Die Kokosnuß kann unser Coconut Joe nur im Urwald finden. Diese Kokosnuß muß Joe aufnehmen, um die Chance zu bekommen, einen Schatz bergen zu können. Nimmt Joe die Kokosnuß auf, so erhält er 10 Punkte auf dem Punktekonto gutgeschrieben. Ferner erhöht sich die Chance, einen Schatz bergen zu können, um 1 Punkt. Der Überlebensbonus wird hierdurch nicht erhöht. Die Kokosnuß wird aufgenommen, indem sie berührt wird. Wenn Coconut Joe die Kokosnuß jedoch überspringt, so werden vom Überlebensbonus 500 Punkte abgezogen, was beim nächsten Szenenwechsel zum Tode führen kann, falls der Lebensbonus auf 0 gefallen ist.

Kaktus
Der Kaktus kommt nur in der Wüste vor. Wenn er außer bei einem Sprung berührt wird, so stirbt unser Joe den Heldentod.
Wird der Kaktus jedoch übersprungen, so werden dem Punktekonto 40 Punkte gutgeschrieben und der Überlebensbonus erhöht sich um 100 Punkte.

Wasserstelle
Die Wasserstelle findet unser Joe ebenfalls nur in der Wüste. Hierfür gelten die gleichen Bedingungen wie für die Kokosnuß, nur daß beim Überspringen 1000 Punkte vom Überlebensbonus abgezogen werden.



Goldbarren
Den Goldbarren findet unser Joe nur in den unterirdischen Katakomben. Wird der Goldbarren berührt, und die Chance einen Schatz bergen zu können ist größer 0, so verschwindet der Goldbarren und dem Punktekonto werden 5000 Punkte gutgeschrieben. Wenn der Goldbarren jedoch berührt wird wenn die Chance 0 ist (d. h. wenn keine Kokosnuß oder Wasserstelle mehr zur Verfügung steht), stirbt Joe und das Spiel ist zu Ende. Wenn ein Goldbarren aufgenommen wird, so verringert sich die Chance um 1. Es kann also für jede angesammelte Kokosnuß oder Wasserstelle ein Schatz geborgen werden.

Silberbarren
Für den Silberbarren gelten die gleichen Bedingungen wie für den Goldbarren, nur gibt es beim Einsammeln 3000 Punkte auf dem Punktekonto.
Weiter gibt es im Spielgeschehen noch andere Hindernisse, so zum Beispiel die Fallgruben. Wenn Joe über eine Fallgrube läuft, ohne zu springen, so fällt er hinein und das Spiel ist zu Ende.
Wenn in einer Fallgrube eine Leiter steht, so kann Joe in die Katakomben gehen. Dazu muß der Joystick jedoch beim normalen Lauf schräg nach unten gezogen werden, da wenn Joe gerade über eine Fallgrube mit Leiter geht, er abstürzt und das Spiel zu Ende ist. In den unterirdischen Katakomben stehen auch Mauern, welche den Durchgang versperren. Wenn Joe eine Mauer berührt, so stirbt er, und das Spiel ist zu Ende. Coconut Joe hat schließlich nur ein einziges Leben.
Bei jedem Szenenwechsel wird der Überlebensbonus um 100 Punkte verringert. Wenn in diesem Moment der Bonus auf 0 steht, so ist das Spiel zu Ende. Ein Szenenwechsel verläuft immer dann, wenn Joe den rechten oder linken Bildschirmrand erreicht. Verläßt Joe den Bildschirm rechts, so erscheint er in der nächsten Szene links. Verläßt er den Bildschirm links, so erscheint er in der nächsten Szene rechts. Die Szenen werden durch den Zufall ausgewählt.

Die Anzeige
Die Anzeige ist am oberen Bildschirmrand eingeblendet. Links wird die Punktzahl angezeigt, in der Mitte ist die Chance zu sehen, welche anzeigt, ob und wieviele Schätze geborgen werden können.
Rechts ist der Überlebensbonus zu sehen. Bei Spielbeginn hat dieser Bonus den Wert von 2000 Punkten. Bei jedem Szenenwechsel verringert sich dieser Wert um 100 Punkte.
Die Punktzahl des Überlebensbonus kann nur in der oberen Ebene aufgefrischt werden. Seien Sie deshalb mit einem Szenenwechsel in den Katakomben vorsichtig, da man nie weiß, wann man wieder über eine Leiter ans Tageslicht gelangen kann. Man kann durch den Überlebensbonus in den Katakomben viel schneller das Leben verlieren als an der Oberfläche.
Das Spiel läuft zu Beginn nur im Urwald ab. Erst wenn Sie mindestens 50 000 Punkte erreicht haben, kommen Sie in die Wüste. Das ist kein einfaches Unterfangen. Das Spiel geht dann freilich nur noch in der Wüste weiter.
Doch jetzt genug der Erklärungen. Ich wünsche ein ERRORfreies Eintippen und viel Spaß beim Spiel. Nachfolgend noch einige Tips zum Eintippen:

1. Einige Zeilen sind länger als es der TI zuläßt. Tippen Sie die Zeile ein und schicken Sie diese mit <ENTER> ab. Danach holen Sie die Zeile mit <FCTN><REDO> wieder zurück und können den Rest eingeben.
2. Benutzer eines Diskettenlaufwerkes ohne Speichererweiterung sollten das Spiel bei abgeschalteter Peripheriebox eintippen und über einen Cassettenrecorder abspeichern, da durch den Dateibuffer des Laufwerkes der Speicher beim Programmlauf zu klein ist.
3. Wenn Sie ein Laufwerk mit Erweiterung besitzen, so ist vor dem Einlesen des Programmes von Cassette die Befehlsfolge: CALL FILES(1) <ENTER> NEW <ENTER> einzugeben, damit das Programm von Cassette eingelesen werden kann.
4. Während des Spielablaufes darf die <ALPHA LOCK> Taste nicht eingerastet sein, da sonst der Joystick nicht richtig abgefragt wird.
5. Tippen Sie vor allem die DATA Zeilen sehr genau ab, da ein hieraus resultierender Fehler sehr schwer zu finden ist.

Die genaue Programmstruktur ist der beigelegten Zuordnungstafel zu entnehmen. Nachfolgend wird aber zusätzlich noch eine Kurzbeschreibung des Programmes gegeben:

Zeile	Programmeinheit
1– 11	Copyright Zeilen
20– 133	Programmvorbereitung
135– 136	Aufruf des Titelbildes
137	Sprung ins eigentliche Spielprogramm
140– 170	Zufällige Auswahl einer der 378 verschiedenen Spielszenen
1000– 1110	Coconut Joe in der oberen Ebene
1500– 1520	Szenenwechsel in der oberen Ebene
2000– 2002	Kontrolle, ob Joe eine Leiter betritt
2005– 2010	Spielende durch Tod von Coconut Joe
3000– 3200	Kontrolle und Verzweigung, wenn Joe einen Gegenstand berührt
4000– 4030	Sprung von Coconut Joe
4500– 4530	Coconut Joe auf der Leiter
4550– 4640	Coconut Joe in der unteren Ebene
4800– 4820	Szenenwechsel in den Katakomben
4900	Schatz aufnehmen
4990– 4995	Betreten der Leiter in den Katakomben
5000– 5020	Sprung von der Leiter in die obere Ebene
6500– 6530	Abkontrolle des Titelbildes und Initialisierung der restlichen Grafik
10000–25030	Zusammenstellung der einzelnen Szenen
26000–26030	Punktanzeige
26040–26070	Anzeige der Chancen einen Schatz zu heben
26080–26110	Anzeige des Überlebensbonus
27000–27080	Ausgabe des Gonges bei Einsammeln eines Gegenstandes
28000–28014	Copyright
29900–29904	Abspeichern der Charakteren 48 bis 57
30000–30140	Bild ausgeben
30510–30575	Anzeige der Titelschrift
30800–30845	Generierung der Grafik Coconut Joe
31000–31160	Spielen des Titelliedes
31500–31540	Melodie vor Spielende
32000–32110	Belegung der Hauptgrafik

Wenn Coconut Joe gestorben ist, so erscheint wieder das Titelbild. Durch Druck auf den Aktionsknopf können Sie ein neues Spiel beginnen.
Ich hoffe, daß Sie genausoviel Spaß beim Spielen dieses abwechslungsreichen Spieles haben werden wie ich es habe und auch schon bei der Programmierung hatte.
ANMERKUNG: Pro Szene werden die oben genannten Punkte nur einmal vergeben, auch wenn der betreffende Gegenstand mehrmals übersprungen wird.
Wem es zu lange dauert, in die Wüste zu kommen, der kann die erforderliche Punktzahl in Zeile 142 entsprechend reduzieren.

```
1 REM  ***************************************************
2 REM  *                    M A Y A                      *
3 REM  *-------------------------------------------------*
4 REM  * Ein Spiel in 378 Bildern fuer den TI 99/4 A     *
5 REM  * geschrieben in Extended BASIC fuer Joysticks    *
6 REM  *-------------------------------------------------*
7 REM  * (c) 1984 by Volker Becker                       *
8 REM  *     Steinbacher Str.10,  6370 Oberursel 6       *
9 REM  ***************************************************
10 REM
11 REM
20 RANDOMIZE
40 CALL CLEAR :: CALL INITIAL(PAT$()):: CALL ASCIISET
110 CALL MAGNIFY(3):: CALL SCREEN(4):: FOR I=1 TO 14 :: CALL COLOR(I,1,1):: NEXT
 I
130 CALL BILD :: CALL CRIGHT(PAT$())
132 GOSUB 10000 :: GOSUB 11100 :: GOSUB 12000 :: CALL DELSPRITE(ALL)
133 P=0 :: M=0 :: Z=2100
135 CALL COLOR(3,2,13,4,2,13,12,11,7):: CALL TITEL
```

```
136 GOSUB 6500
137 GOTO 1000
140 CALL DELSPRITE(ALL)
141 SKP=0 :: Z=Z-100 :: CALL ZEIT(Z):: CALL PUNKTE(P):: CALL MOEGLICHKEIT(M):: I
F Z=0 THEN CALL KAMERAD :: GOTO 132
142 IF P<50000 THEN 145 ELSE GOSUB 10200 :: GOTO 150
145 ON INT(RND+1.5)GOSUB 10000,10100
150 ON INT(RND*2+1.5)GOSUB 11000,11100,11200
160 ON INT(RND*2+1.5)GOSUB 12000,12100,12200
165 IF INT(RND*19+1.5)=5 THEN CALL SPRITE(#3,36,9,176,85):: P2=5000 :: VAR=0 ELS
E IF INT(RND*19+1.5)=17 THEN CALL SPRITE(#3,36,15,176,157):: P2=3000 :: VAR=2
170 RETURN
1000 GOSUB 140
1010 CALL SPRITE(#1,68,13,122,230)
1020 CALL JOYST(1,X,Y):: IF X=0 THEN 1040
1030 CALL SOUND(-5,523,0):: IF X<0 THEN CALL PATTERN(#1,68)ELSE CALL PATTERN(#1,
84)
1040 CALL MOTION(#1,0,X*2)
1045 CALL KEY(1,T,S):: IF T=18 THEN 4000
1050 CALL POSITION(#1,X,Y):: IF Y<12 OR Y>239 THEN 1500
1060 CALL GCHAR(18,(Y+9)/8,BV):: IF BV=BVO THEN 2000
1070 CALL JOYST(1,X,Y):: IF X=0 THEN 1090
1080 CALL SOUND(-5,261,0):: IF X<0 THEN CALL PATTERN(#1,76)ELSE CALL PATTERN(#1,
92)
1090 CALL MOTION(#1,0,X*2)
1095 CALL KEY(1,T,S):: IF T=18 THEN 4000
1100 CALL COINC(ALL,ST):: IF ST THEN 3000
1110 GOTO 1020
1500 GOSUB 140
1510 IF Y>239 THEN CALL SPRITE(#1,84,13,122,12)ELSE CALL SPRITE(#1,68,13,122,230
)
1520 GOTO 1020
2000 CALL GCHAR(20,(Y+9)/8,BV):: IF BV>135 THEN CALL JOYST(1,X,Y):: GOTO 2002 EL
SE 2005
2002 IF Y=-4 THEN 4500
2005 CALL MOTION(#1,0,0):: CALL PATTERN(#1,132):: CALL MOTION(#1,10,0)
2010 CALL KAMERAD :: GOTO 132
3000 CALL MOTION(#1,0,0)
3005 ON SP2 GOTO 3100,3200
3100 GOTO 2005
3200 M=M+1 :: CALL MOEGLICHKEIT(M):: CALL DELSPRITE(#2):: P=P+10 :: CALL PUNKTE(
P):: CALL GONG(1):: GOTO 1020
4000 STV=0 :: CALL POSITION(#1,X,Y):: CALL LOCATE(#1,112,Y)
4010 FOR I=110 TO 190 STEP 10 :: CALL SOUND(-100,I,0,I+2,0,I+4,0):: CALL COINC(A
LL,ST):: IF ST THEN STV=1
4015 NEXT I
4020 CALL POSITION(#1,X,Y):: CALL LOCATE(#1,122,Y)
4025 IF STV=1 AND SKP=0 THEN P=P+P1 :: CALL PUNKTE(P):: Z=Z+Z1 :: CALL ZEIT(Z)::
 SKP=SKP+1
4030 GOTO 1020
4500 CALL PATTERN(#1,132):: CALL MOTION(#1,0,0):: CALL LOCATE(#1,138,121)
4510 CALL POSITION(#1,X1,Y):: IF X1>=173 THEN 4550 ELSE IF X1<=137 THEN 5000
4520 CALL JOYST(1,X,Y):: CALL MOTION(#1,-Y*1.5,0):: IF Y=0 THEN 4530 ELSE CALL S
OUND(-50,523-X1,0)
4530 GOTO 4510
4550 CALL JOYST(1,X1,Y):: IF X=0 THEN 4570
4560 CALL SOUND(-5,523,0):: IF X1<0 THEN CALL PATTERN(#1,68)ELSE CALL PATTERN(#1
,84)
4570 CALL MOTION(#1,0,X1*2)
4580 CALL POSITION(#1,X,Y):: IF Y<12 OR Y>239 THEN 4800
4590 CALL GCHAR(21,(Y+9)/8,BV):: IF BV=BVU THEN 2005 ELSE IF BV>135 THEN 4990
```

```
4600 CALL JOYST(1,X,Y):: IF X=0 THEN 4620
4610 CALL SOUND(-5,261,0):: IF X<0 THEN CALL PATTERN(#1,76)ELSE CALL PATTERN(#1,
92)
4620 CALL MOTION(#1,0,X*2)
4630 CALL COINC(ALL,ST):: IF ST THEN 4900
4640 GOTO 4550
4800 GOSUB 140
4810 IF Y>239 THEN CALL SPRITE(#1,84,13,176,12)ELSE CALL SPRITE(#1,68,13,176,230
)
4820 GOTO 4550
4900 IF M>0 THEN CALL DELSPRITE(#3):: P=P+P2 :: CALL PUNKTE(P):: M=M-1 :: CALL M
OEGLICHKEIT(M):: CALL GONG(VAR):: GOTO 4550 :: ELSE 2005
4990 IF BVO<>81 THEN 4600
4992 CALL JOYST(1,X,Y):: IF Y=4 THEN 4995 ELSE 4600
4995 CALL PATTERN(#1,132):: CALL MOTION(#1,0,0):: CALL LOCATE(#1,172,121):: GOTO
 4510
5000 CALL MOTION(#1,0,0):: CALL JOYST(1,X,Y):: IF Y=-4 THEN 4520 ELSE IF X=0 THE
N 5000 ELSE 5010
5010 CALL LOCATE(#1,122,121+4*X):: IF X<0 THEN CALL PATTERN(#1,68)ELSE CALL PATT
ERN(#1,84)
5020 GOTO 1020
6500 CALL CONDORPASA(B(),1)
6505 CALL KEY(1,T,S):: IF T<>18 THEN 6505
6510 CALL ASCII2 :: CALL HCHAR(2,1,58,32)
6520 FOR I=48 TO 57 :: CALL CHAR(I,PAT$(I-48)):: NEXT I
6530 RETURN
10000 CALL COLOR(3,2,13,4,2,13,1,1,1,2,13,1,5,7,1,6,1,1):: CALL SCREEN(4):: GOSU
B 20000 :: RETURN ! 2 Baumstaemme
10100 CALL COLOR(3,2,13,4,2,13,1,1,1,2,13,1,5,1,1,6,7,1):: CALL SCREEN(4):: GOSU
B 20000 :: RETURN ! 1 Baumstamm
10200 CALL COLOR(3,2,12,4,2,12,1,11,1,2,1,1,5,1,1,6,1,1):: CALL SCREEN(12):: GOS
UB 25000 :: RETURN ! Pyramiden
11000 CALL COLOR(9,7,2,10,2,2):: BVU=96 :: RETURN ! linke Mauer
11100 CALL COLOR(9,2,2,10,7,2):: BVU=104 :: RETURN ! rechte Mauer
11200 CALL COLOR(9,2,2,10,2,2):: BVU=0 :: RETURN ! keine Mauer
12000 CALL COLOR(14,8,2,7,1,2,8,7,2,11,7,7,13,11,11):: BVO=81 :: RETURN ! 1 Fall
grube
12100 CALL COLOR(14,2,2,7,11,11,8,7,7,11,7,2,13,2,2):: BVO=129 :: RETURN ! 2 Fal
lgruben
12200 CALL COLOR(14,2,2,7,11,11,8,7,7,11,7,7,13,11,11):: BVO=0 :: RETURN ! keine
 Fallgrube
20000 ON INT(RND*4+1.5)GOTO 20100,20200,20300,20400,20500
20100 RETURN
20200 CALL SPRITE(#2,116,7,122,157,0,-2):: SP2=1 :: P1=100 :: Z1=300 :: RETURN !
 Schlange
20300 CALL SPRITE(#2,100,7,122,20,0,10):: SP2=1 :: P1=50 :: Z1=200 :: RETURN !rn"
llender Baumstamm
20400 CALL SPRITE(#2,60,12,122,85):: SP2=1 :: P1=30 :: Z1=100 :: RETURN ! Feuer
20500 CALL SPRITE(#2,124,7,122,157):: SP2=2 :: P1=0 :: Z1=-500 :: RETURN ! Kokos
nuss
25000 ON INT(RND*3+1.5)GOTO 20100,25020,20200,25030
25020 CALL SPRITE(#2,108,5,122,85):: SP2=1 :: P1=40 :: Z1=100 :: RETURN ! Kaktus
25030 CALL SPRITE(#2,140,8,122,157):: SP2=2 :: P1=0 :: Z1=-1000 :: RETURN ! Wass
erstelle
26000 SUB PUNKTE(P)
26010 A$=STR$(P)
26020 DISPLAY AT(1,1)SIZE(LEN(A$)+3):A$&":::"
26030 SUBEND
26040 SUB MOEGLICHKEIT(M)
26045 IF M>99 THEN M=99
26050 A$=STR$(M)
```

```
26060 DISPLAY AT(1,14)SIZE(LEN(A$)+3):A$&":::"
26070 SUBEND
26080 SUB ZEIT(Z)
26085 IF Z<=0 THEN Z=0
26090 A$=STR$(Z)
26100 DISPLAY AT(1,20)SIZE(LEN(A$)+4):A$&"::::"
26110 SUBEND
27000 SUB GONG(VAR)
27005 CALL MOTION(#1,0,0)
27010 FOR I=0 TO 9 :: CALL SOUND(-250,391,I):: NEXT I
27020 IF VAR=1 THEN SUBEXIT
27030 FOR I=0 TO 9 :: CALL SOUND(-250,329,I,391,I+11):: NEXT I
27040 FOR I=0 TO 9 :: CALL SOUND(-250,261,I,329,I+11,391,I+21):: NEXT I
27050 IF VAR=2 THEN SUBEXIT
27060 FOR I=0 TO 9 :: CALL SOUND(-250,391,I,261,I+11,329,I+21):: NEXT I
27070 FOR I=0 TO 9 :: CALL SOUND(-250,261,I,391,I+11,329,I+21):: NEXT I
27080 SUBEND
28000 SUB CRIGHT(PAT$())
28002 DATA 48,3C4299A1A199423C,50,66A5242442432418,51,78A42438242 3A27C,53,00001C
24F821120C,54,00003C40C041423C,55,08141838D4151916
28004 DATA 57,0000334C88080808
28006 RESTORE 28002
28008 FOR I=1 TO 7 :: READ C,C$ :: CALL CHAR(C,C$):: NEXT I
28009 CALL CHAR(56,PAT$(8),52,PAT$(4))
28010 DATA 48,58,56,52,58,50,58,51,53,54,55,53,57
28012 FOR I=1 TO 13 :: READ C :: CALL HCHAR(2,I+9,C):: NEXT I
28014 SUBEND
29900 SUB INITIAL(PAT$())
29902 FOR I=48 TO 57 :: CALL CHARPAT(I,PAT$(I-48)):: NEXT I
29904 SUBEND
30000 SUB BILD
30010 CALL HCHAR(1,1,58,64):: CALL HCHAR(20,1,59,160)
30020 DISPLAY AT(3,1):"(........)(.......)(........) (......)  (....)  (......)
  (..)      (..)      (..)        ,-        ,-        ,-"
30030 CALL VCHAR(7,7,65,9):: CALL VCHAR(7,8,66,9):: CALL VCHAR(7,16,73,9):: CALL
 VCHAR(7,17,74,9)
30040 CALL VCHAR(7,25,65,9):: CALL VCHAR(7,26,66,9)
30050 CALL HCHAR(16,1,120,96):: CALL HCHAR(19,1,121,32)
30060 DATA 7,129,112,25,129,112,16,81,89
30070 RESTORE 30060
30080 FOR I=1 TO 3 :: READ S,C1,C2
30090 CALL HCHAR(17,S,C1,2):: CALL HCHAR(18,S,C1,2):: CALL HCHAR(19,S,C2,2):: NE
XT I
30100 CALL VCHAR(20,5,96,5):: CALL VCHAR(20,28,104,5):: CALL VCHAR(20,16,136,5)
:: CALL VCHAR(20,17,137,5)
30110 DATA 7,12,9,20,10,3,11,29,12,11
30120 FOR I=1 TO 5 :: READ Z,S :: CALL HCHAR(Z,S,33):: CALL HCHAR(Z,S+1,34):: CA
LL HCHAR(Z+1,S-1,33):: CALL HCHAR(Z+1,S,35,2):: CALL HCHAR(Z+1,S+2,34)
30130 CALL HCHAR(Z+2,S-2,33):: CALL HCHAR(Z+2,S-1,35,4):: CALL HCHAR(Z+2,S+3,34)
:: NEXT I
30140 SUBEND
30510 SUB TITEL
30520 CALL MAGNIFY(4)
30525 DATA ,0103060C183060C,0103070F1B3363C3,83,0103070F1B3363C3,83030303030303030
3,0F0F18303030303,303F3F303030303,E1E1311919191818
30530 DATA 18F8F818181818,8C8C8C8C8C8CD8D8,707070707070707,80C0E0F0D8CCC6C3,C1
C0FFFFC0C0C0C,,80C0E0F0180C0603
30532 RESTORE 30525
30535 FOR I=67 TO 91 STEP 8 :: FOR K=1 TO 4
30540 READ A$ :: CALL CHAR(I+K,A$)
30545 NEXT K :: NEXT I
```

```
30550 J=64
30555 DATA 68,76,84,92
30560 FOR I=1 TO 4
30565 READ A :: CALL SPRITE(#I,A,5,89,J):: J=J+32
30570 NEXT I
30575 SUBEND
30800 SUB ASCII2
30805 CALL DELSPRITE(ALL)
30810 CALL MAGNIFY(3)
30815 DATA 68,0001000000C0201,69,0103030301020C,70,E0E0A0E0202060E,71,E0E0E0E0404
040C,76,0001000000000708,77,0103030301010103
30820 DATA 78,E0E0A0E02020E0E,79,E0E0E0E04020101,84,0707050704040607,85,07070707
02020203,86,008000003040 8
30825 DATA 87,80C0C0C080403,92,0707050704040707,93,0707070702040808,94,008000000
000F01,95,80C0C0C0808080C
30829 RESTORE 30815
30830 FOR I=1 TO 16
30835 READ C,C$ :: CALL CHAR(C,C$)
30840 NEXT I
30845 SUBEND
31000 SUB CONDORPASA(B(),KY)
31010 DATA 2,30000,30000,30000,30000,1,15000,293,1,15000,277,1,15000,293,1,15000
,329,1,15000,349,1,15000,329
31020 DATA 1,15000,349,1,15000,391,6,220,261,349,440,6,349,440,2,440,523,6,349,4
40,1,466,587,1,440,523
31030 DATA 6,349,440,1,15000,391,1,15000,349,4,174,220,293,349
31040 DATA 6,15000,293,2,15000,349,7,15000,293,1,15000,220
31041 DATA 2,30000,30000,30000,30000,1,15000,293,1,15000,277,1,15000,293,1,15000
,329,1,15000,349,1,15000,329
31042 DATA 1,15000,349,1,15000,391,6,220,261,349,440,6,349,440,2,440,523,6,349,4
40,1,466,587,1,440,523
31043 DATA 6,349,440,1,15000,391,1,15000,349,4,174,220,293,349
31044 DATA 6,15000,293,2,15000,349,7,15000,293,1,15000,440,4,15000,15000,15000,1
5000,3,466,587,1,440,554
31045 DATA 1,466,587,1,440,554,1,466,587,1,440,554,6,466,587,1,587,698,1,466,587
,4,220,261,349,440
31046 DATA 6,440,523,1,466,587,1,440,523,6,349,440,1,15000,391,1,15000,349,4,174
,220,293,349,6,15000,293
31047 DATA 2,15000,349,8,15000,293,0
31060 RESTORE 31010
31070 AZ,STV=0
31075 FOR I=0 TO 2 :: CALL SOUND(-600,220,I):: NEXT I
31080 READ K :: IF K=0 THEN 31060
31090 READ B(1),B(2),B(3),B(4)
31100 FOR I=1 TO K
31110 FOR J=1 TO 4
31120 STV=STV+1 :: IF STV>AZ THEN READ AZ,M1,M2 :: STV=1
31130 FOR L=0 TO 2 :: CALL SOUND(-600,M1,L+1,M2,L,B(J),9):: NEXT L :: CALL KEY(1
,T,S):: IF S<>0 THEN SUBEXIT
31140 NEXT J :: NEXT I
31150 GOTO 31080
31160 SUBEND
31500 SUB KAMERAD
31510 DATA 500,130,500,174,250,220,25,30000,100,220,25,30000,500,220,500,195,100
0,174,500,130
31520 RESTORE 31510
31525 CALL DELSPRITE(#1)
31530 FOR I=1 TO 10 :: READ L,T :: CALL SOUND(L,T,0):: NEXT I
31540 SUBEND
32000 SUB ASCIISET
32010 DATA ,0103070F1F3F7FFF,80C0E0F0F8FCFEFF,FFFFFFFFFFFFFFFF,0000000103070F1D,
3E7DFFD7ABFE,007FFFD7AF96BC38,B060C08,FFFF7F7F3F1F0701
```

```
32020 DATA FFFFFFFFFCF8E08,030F1F3F3F7F7FFF,C0F0F8FCFCFEFEFF,FFFF7F7F3F3F1F1F,FF
FFFFFFFCFCF8F8,FFFFFFFFFFFFFFFF,,,,,,,,,,,,,FFFFFFFFFFFFFFFF
32030 DATA 00004163567D2B1E,0F07030003060C18,0002068ADE74B8F,E0C080008C0603,,0F
0F1F1F0F0F0F0F,F0F0F0F0F8F8F8F,,,,,,,0F0F1F1F0F0F0F0F
32040 DATA F0F0F0F0F8F8F8F0,,,,,,,,,,,,,,FFFF,,,,,,,FBFBFB00DFDFDF,,,,000000000
103070A,153F4792BB924438,1C2E5FFBD7AEFCF8,F0A0408
32050 DATA FBFBFB00DFDFDF,,,,030303030318181B,1B1B1F0F03030303,00000C0C6C6C6C7C,
786060606060E0C,FFFF,,,,000E1B2E4,050A10150A000A05
32060 DATA 0000806030081008,50A00050A010A040,FFFFFFFFFFFFFFFF,,,,0000000000001030
6,070D0D070603030 1,0000000000080C060,E0B0B0E060C0C08
32070 DATA FFFFFFFFFFFFFFFF,,,,0003070363110907,03070F1F02040818,0080C0808C1020C
,80C0F0F08040203,1F1F18181F1F1818,F8F81818F8F81818
32080 DATA ,,0000000000030F3FFF,FF3F0F03,0000000000C0F0FCFF,FFFCF0C
32090 RESTORE 32010
32100 FOR I=32 TO 143 :: READ C$ :: CALL CHAR(I,C$):: NEXT I
32110 SUBEND
```

ASC II – DEF

Definition von Charakteren, Teil 2
Umwandeln eines Codefeldes

Wenn Sie eine Datei mittels des Programmes ASCIIDEF oder des Unterprogrammes ACIIPAT erstellt haben, so muß diese Datei in einen für den Drucker lesbaren Grafikcode umgesetzt werden. Das ist so zu verstehen, daß für den Drucker ein zweiter Zeichensatz generiert wird. Dazu wird das Programm ASCIIUMW verwendet. Das Programm arbeitet nach folgendem Prinzip: Der im Codefeld enthaltene Hexadezimale Code wird in einen Dezimalen Code umgesetzt. Mittels dieses Dezimalcodes wird intern eine binäre 8 x 8 Punkte Matrix erstellt. Diese Matrix wird dann von Spalte 8 bis Spalte 1 von Zeile 1 bis 8 abgerastert und von jeder Spalte der dazugehörige Dezimalcode ermittelt. Die dem jeweiligen Dezimalcode zugeordneten Zeichen werden jetzt in einer Stringverknüpfung zusammengesetzt. So entsteht für jedes Zeichen intern ein aus 8 Zeichen bestehender String. Dieser String wird auf der Datei zusätzlich zu dem Hexadezimalcode abgespeichert und entspricht im Grafikmodus des Druckers exakt dem auf dem Bildschirm zugeordneten Zeichen.

Laden Sie das Programm ASCIIUMW ein und starten Sie dieses mit RUN. Sie werden nun aufgefordert den Namen des entsprechenden Codefeldes (z. B. DSK1.FELD oder CS1) einzutippen. Alles weitere erledigt der Rechner für Sie. Das umgewandelte Codefeld wird wieder unter dem gleichen Namen auf den entsprechenden Datenträger zurückgeschrieben. Auch ein umgewandeltes Codefeld kann mittels des Programmes ASCIIDEF eingelesen und editiert werden, muß jedoch wenn es zurückgeschrieben wird, erneut umgewandelt werden. Die Umwandlung eines Codefeldes muß in der Regel nur einmal erfolgen um den Zeichensatz für den Drucker zu belegen. Die Umwandlung dauert mittels Diskette etwa 25 Minuten, mit Recorder dagegen ca. 1 Stunde. Der Rechner informiert jedoch laufend, welches Zeichen im Moment umgewandelt wird. Die so umgewandelte Datei ist Voraussetzung für den Ausdruck des Zeichensatzes sowie für die Erstellung einer Hardcopy. Auch mit einer umgewandelten Datei kann mittels des Unterprogrammes ASCIISET der Zeichensatz umbelegt werden, da der Hexadezimalcode nicht zerstört wird.

Ausdruck des grafischen Zeichensatzes

Um den Zeichensatz auszudrucken, wird das Programm ASCIIPRINT herangezogen. Laden Sie das Programm ASCIIPRINT ein und starten Sie dieses mit RUN. Sie werden aufgefordert den Namen des Codefeldes einzugeben (z. B. DSK1.FELD oder CS1). Danach werden Sie nach dem Programmnamen gefragt, der in der Seitenüberschrift erscheinen soll. Nach Eingabe dieser Informationen erfolgt der Ausdruck. Achten Sie jedoch darauf, daß der Drucker vor dem Programmstart am Anfang einer neuen Seite steht, da das Programm den Seitenvorschub von 72 Zeilen beim Drucker festlegt, kurz nachdem es gestartet wurde.

Erstellen einer Hardcopy

Zuvor müssen einige Einschränkungen gemacht werden. Das Programm erstellt eine Hardcopy des Bildschirmes mit einem selbst belegten grafischen Zeichensatz. Dazu muß jedoch ein umgewandeltes Codefeld vorhanden sein. Eine Hardcopy wird nur von der auf dem Bildschirm erscheinenden Grafik erstellt, nicht jedoch von Sprites. Die Spritecharakteren müssen also, wenn sie nicht copiert werden sollen, mit CALL HCHAR oder CALL VCHAR vorher als normale Grafik auf den Bildschirm gebracht werden. Das Programm copiert den Bildschirm von Zeile 1 bis 24 und von Spalte 2 bis 31. Spalte 1 und 32 werden nicht copiert, sind jedoch auf dem Bildschirm meist auch nicht sichtbar, da sie die linke und rechte Begrenzung darstellen.

Doch nun zum eigentlichen Copiervorgang: Tippen Sie das Unterprogramm COPY zusätzlich zu Ihrem normalen Programm ein oder laden Sie dieses mit MERGE auf ein bestehendes Programm auf. Rufen Sie das Unterprogramm an gegebener Stelle im Programm auf mit Angabe der entsprechenden Datei, auf welche der Bildschirm gespeichert werden soll. So z. B. CALL COPY („DSK1.BILD") oder CALL COPY („CS1"). Bei der Verwendung eines Recorders ist jedoch Vorsicht geboten, da der Rechner die Steueranweisung auf dem Bildschirm ausgibt, und somit das zu copierende Bild scrollt. Es ist deshalb bei Recorderbetrieb besser den OPEN Befehl aus dem Unterprogramm herauszunehmen und OPEN #88:„CS1" ... in das Hauptprogramm vor die Generierung des Bildes zu schreiben. Das Unterprogramm heißt dann nur noch COPY und wird mit CALL COPY aufgerufen. Die so gespeicherten Copien können nur mit dem Programm COPYPRINT umgewandelt werden bzw. ausgedruckt werden.

Laden Sie zu diesem Zweck das Programm COPYPRINT und starten Sie dieses mit RUN. Sie erhalten dann ein Menü worin Sie auswählen können ob Sie eine Copie umwandeln oder ausdrucken wollen. Das Ausdrucken ist nur mit einem Diskettenlaufwerk möglich, da ein Recorder durch die Struktur des Rechners nur eine Satzlänge von 192 Bytes verarbeiten kann, jedoch eine umgewandelte Hardcopy 24 Sätze zu je 240 Bytes hat. Der Ausdruck einer Hardcopy erfolgt jedoch auch bei der Umwandlung. Da die eigentliche Umwandlung nur ca. 1½ Minuten dauert, können Benutzer eines Recorders über die Umwandlung die Copien ausdrucken. Wenn Sie die Umwandlung gewählt haben, so geben Sie die Datei ein, in welcher der Zei-

chensatz für den Drucker gespeichert ist. Der Rechner liest nun die Datei ein. Danach geben Sie die Datei ein, unter welcher Sie den Bildschirm abgespeichert haben. Der Rechner wandelt jetzt die Bildschirmcopie in ein auszudruckendes Bild um. Dazu wird der 8stellige Druckerstring für das entsprechende Zeichen gelesen, und diese Strings zusammengesetzt. Eine Bildschirmzeile besteht somit aus einem 240 Zeichen langen String. Nach der Umwandlung können Sie, sofern Sie ein Laufwerk besitzen, diese umgewandelte Copie abspeichern. Die Copie wird jedoch ausgedruckt, egal ob Sie abspeichern oder nicht. Wenn Sie weitere Umwandlungen mit dem gleichen Zeichensatz wünschen, so antworten Sie bei der Frage mit ‚J'. Dann geht es weiter mit dem Einlesen der Bildschirmcopie.

Dieses Programmpaket soll einige Anregungen bringen, was mit dem TI in EX-BASIC alles machbar ist. Einige Programme können ohne weiteres zusammengefaßt oder umgeändert werden, so daß z. B. eine Hardcopy direkt im Programm ausgedruckt werden könnte. Dies würde jedoch bei eigenen größeren Programmen eine Speichererweiterung voraussetzen. Disketten-laufwerkbesitzer können auch anstatt der Sequentiellen Dateien Direktzugriffsdateien verwenden. Es wurden jedoch Sequentielle Dateien benutzt, da nur diese mit einem Recorder benutzt werden können, und das Programmpaket so vielen Benutzern wie möglich von Nutzen sei soll. Programmintern werden die Dateien sowieso als Direktzugriffsdateien benutzt. Dazu wird schließlich die komplette Datei in Felder eingelesen.

Ich würde mich freuen, wenn ich Ihnen die mühselige Arbeit der Zeichendefinition mit diesen Programmen erleichtern könnte.

```
10 REM ******************************
12 REM *      A S C I I U M W       *
14 REM * Umwandeln eines Codefeldes *
16 REM *----------------------------*
18 REM *   TI 99/4 A                *
20 REM * + TI Extended BASIC        *
22 REM * + externer Speicher        *
24 REM *----------------------------*
26 REM * (c) 1984 by Volker Becker  *
28 REM *     Steinbacher Str. 10    *
30 REM *     6370 Oberursel 6       *
32 REM ******************************
34 REM
36 REM
100 DIM ASCII$(112),G$(112)
110 DISPLAY AT(1,1)ERASE ALL:"UMWANDLUNG EINES CODEFELDES       ZUR GRAPHISCHEN
            DRUCKER-AUSGABE        ----------------------------"
120 DISPLAY AT(6,1):"NAME DES UMZUWANDELNDEN     CODEFELDES (GERAET.NAME)"
130 ACCEPT AT(8,1)BEEP:DATEI$
140 ON ERROR 130
150 OPEN #1:DATEI$,INPUT ,SEQUENTIAL,INTERNAL,FIXED 80
160 ON ERROR STOP
170 FOR I=32 TO 143
180 INPUT #1:G$(I-32),ASCII$(I-32)
190 NEXT I
200 CLOSE #1
210 OPEN #1:DATEI$,OUTPUT,SEQUENTIAL,INTERNAL,FIXED 80
220 DISPLAY AT(10,1):"UMWANDLUNG DES ZEICHENS->"
230 FOR K=32 TO 143
240 DISPLAY AT(10,26):USING "###":K
250 HEX$=ASCII$(K-32)
260 M=1
270 C$=""
280 H$=SEG$(HEX$,1,2)
290 E=0
300 FOR L=LEN(H$)TO 1 STEP -1
310 B$=SEG$(H$,L,1)
320 FOR J=0 TO 5
330 IF ASC(B$)=65+J THEN A=10+J :: GOTO 360
340 NEXT J
350 A=VAL(B$)
360 D=D+A*16^E
370 E=E+1
380 NEXT L
390 IF D=0 THEN FELD$(M)="00000000" :: GOTO 450
400 D=D/2
410 IF D=INT(D)THEN C$="0"&C$ :: GOTO 430
420 C$="1"&C$ :: D=INT(D)
```

```
430 IF INT(D)>0 THEN 400
440 FELD$(M)=C$ :: C$=""
450 M=M+1
460 HEX$=SEG$(HEX$,3,LEN(HEX$)-2)
470 IF HEX$<>"" THEN 280
480 D=0
490 FOR R=1 TO 8
500 IF LEN(FELD$(R))<8 THEN FELD$(R)="0"&FELD$(R):: GOTO 500
510 NEXT R
520 GRAPH$=""
530 FOR N=1 TO 8
540 FOR O=8 TO 1 STEP -1
550 B$=SEG$(FELD$(O),N,1)
560 D=D+VAL(B$)*2^(O-1)
570 NEXT O
580 GRAPH$=GRAPH$&CHR$(D)
590 D=0
600 NEXT N
610 G$(K-32)=GRAPH$
620 NEXT K
630 FOR I=32 TO 143
640 PRINT #1:G$(I-32),ASCII$(I-32)
650 NEXT I
660 CLOSE #1
670 DISPLAY AT(10,1):"NOCH EINE UMWANDLUNG (J/N)"
680 ACCEPT AT(10,28)VALIDATE("JN")SIZE(1)BEEP:SKP$
690 IF SKP$="J" THEN 110
700 CALL CLEAR
710 STOP
720 END
```

```
100 REM ***************************************************
110 REM *              A S C I I P R I N T                *
120 REM * Ausdrucken eines Codefeldes ( ASCII - Satz )    *
130 REM *-------------------------------------------------*
140 REM *   T I  99/4  A                                  *
150 REM * + TI Extended BASIC                             *
160 REM * + externer Speicher (Laufwerk oder Recorder)    *
170 REM * + Seikosha GP 700 A (oder anderen kompartiblen*
180 REM *   Drucker mit 8 DOT Matrix                      *
190 REM *-------------------------------------------------*
200 REM * (c) 1984 by Volker Becker                       *
210 REM *     Steinbacher Str. 10                         *
220 REM *     6370 Oberursel 6                            *
230 REM ***************************************************
240 REM
250 REM
260 REM Druckeroptionen
270 OPEN #99:"PIO" ! Eroeffnen der Paralellen Schnittstelle
280 N$=CHR$(27)&"A" ! Einstellen auf normalen Zeilenabstand
290 SL$=CHR$(27)&"ZH" ! Seitenlaenge auf 72 Zeilen einstellen
300 T10$=CHR$(16)&"010" ! Tabulator auf Position 10 setzen
310 T11$=CHR$(16)&"011" ! Tabulator auf Position 11 setzen
320 ZV$=CHR$(10)! Zeilenvorschub
330 SV$=CHR$(12)! Seitenvorschub
340 CR$=CHR$(13)! Wagenruecklauf ohne Zeilenvorschub
350 U$=CHR$(27)&"R"&CHR$(32)&"_" ! Unterstreichen mit dem Zeichen '_' (32 mal)
360 G$=CHR$(27)&"K008" ! Drucker auf 8 Graphikzeichen vorbereiten
370 REM Ende der Druckeroptionen
380 DISPLAY AT(1,1)ERASE ALL:" Ausdruck des ASCII Satzes   ---------------------
----"
390 DISPLAY AT(4,1):"ASCII Eingabedatei :          (Geraet,Dateiname)"
```

```
400 ACCEPT AT(6,1)BEEP:DATEI$
410 ON ERROR 400
420 OPEN #1:DATEI$,INPUT ,SEQUENTIAL,INTERNAL,FIXED 80
430 DISPLAY AT(8,1):"Programmname :" :: ACCEPT AT(8,16)BEEP:PNAME$
440 PRINT #99:N$;SL$
450 PAGE=1 :: GOSUB 540
460 FOR I=32 TO 143 :: INPUT #1:GRAPH$,HEX$
470 PRINT #99:T11$;
480 PRINT #99,USING "###":I;
490 PRINT #99:"   ";HEX$;"      ";G$;GRAPH$
500 IF I-32=56 THEN PAGE=2 :: PRINT #99:SV$ :: GOSUB 540
510 NEXT I
520 CLOSE #1 :: CLOSE #99
530 STOP
540 PRINT #99:T10$;"Zeichensatz des.Programmes ";PNAME$;"   Seite :";PAGE;ZV$;ZV
$
550 PRINT #99:T10$;"ASCII     Hexadezimal    Zeichen";CR$;T10$;U$
560 RETURN
570 END
```

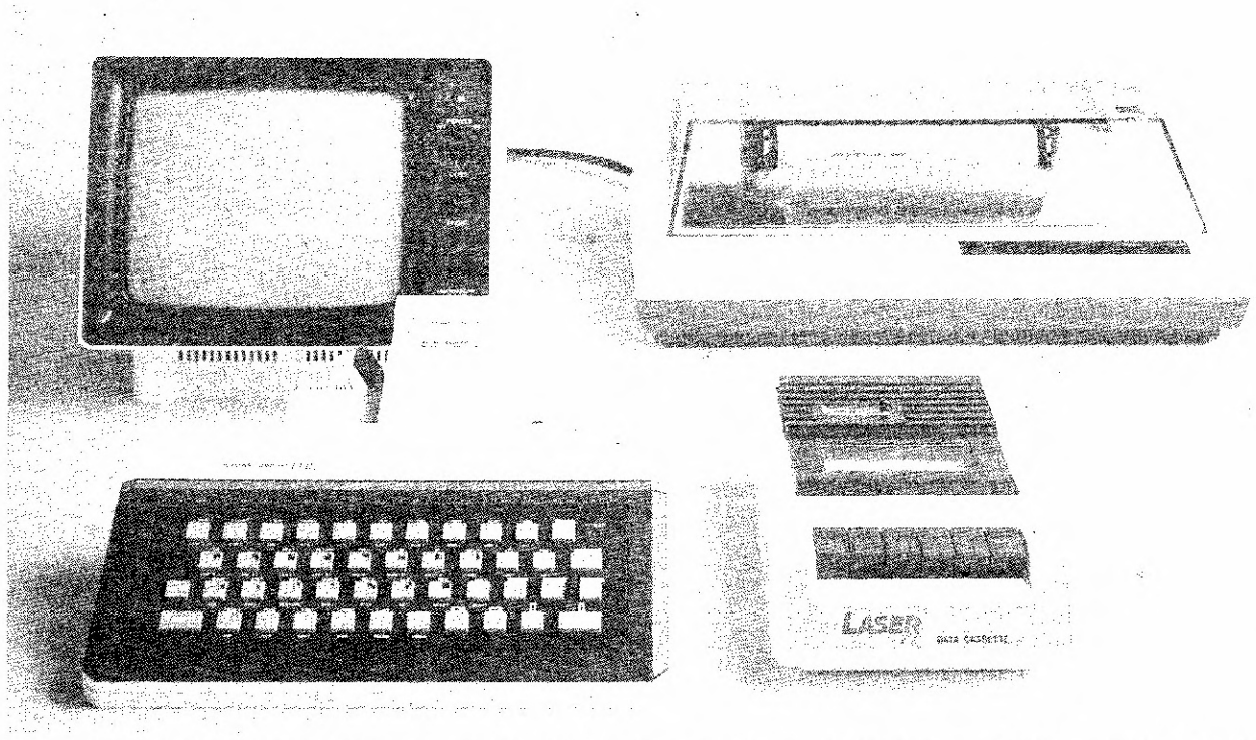
32100 SUB COPY (DATEI)

```
32110 !
32120 ! Unterprogramm zur Abspeicherung des Bildschirmes
32130 ! fuer TI 99/4 A
32140 !
32150 ! (c) 1984 by Volker Becker
32160 !
32170 OPEN #88:DATEI$,OUTPUT,SEQUENTIAL,INTERNAL,FIXED 80
32180 ! VORSICHT - Bei Verwendung eines Cassetenrtrecorders
32190 !            darf der OPEN Befehl nicht im Unterprogramm stehen,
32200 !            sondern muss im Hauptprogramm vor der generierung
32210 !            des Bildschirmes verwendet werden.
32220 !            Die erste Zeile des Unterprogrammes muss dann  heissen SUB CO
PY
32230 !            und es wird nur mit CALL COPY aufgerufen.
32240 !
32250 FOR I=1 TO 24
32255 GRAPH$=""
32260 FOR K=2 TO 31
32270 CALL GCHAR(I,K,CH)
32280 GRAPH$=GRAPH$&CHR$(CH)
32290 NEXT K
32300 PRINT #88:GRAPH$
32310 NEXT I
32315 CLOSE #88
32320 SUBEND
```

```
10 REM ***************************************************
12 REM *                C O P Y P R I N T                *
14 REM * Umwandlung und Ausdruck von Bildschirmcopien    *
16 REM *-------------------------------------------------*
18 REM *   TI 99/4 A + TI Extended BASIC                 *
20 REM * + Externer Speicher (Recorder oder Floppy)      *
22 REM * + Seikosha GP 700 A oder anderen kompartiblen   *
24 REM *   Drucker mit 8 DOT Matrix                      *
26 REM *-------------------------------------------------*
28 REM * (c) 1984 by Volker Becker                       *
30 REM *     Steinbacher Strasse 10                      *
32 REM *     6370 Oberursel 6                            *
34 REM ***************************************************
36 REM
38 REM
40 REM Drucker Optionen
```

```
42 REM ----------------
44 REM
46 TAB$=CHR$(16)&"025" ! Druckkopf auf Blattmitte positionieren
48 G$=CHR$(27)&"K240" ! Drucker auf 240 Graphiksymbole einstellen
50 GS$=CHR$(27)&"T12" ! Zeilenvorschub auf Graphik einstellen
52 N$=CHR$(27)&"A" ! Zeilenvorschub auf Normalmodus einstellen
54 REM
56 REM ----------------
58 REM
90 DIM GRAPH$(112),BILD$(24),COPY$(24)
100 DISPLAY AT(1,1)ERASE ALL:"   Erstellen der Hardcopy   ----------------------
------"
105 GOTO 5000
110 CALL HCHAR(3,1,32,704)
115 DISPLAY AT(3,1):"Name des CODEFELDES          (GERAET.NAME)"
120 ACCEPT AT(5,1)BEEP:DATEI$
130 ON ERROR 120
140 OPEN #1:DATEI$,INPUT ,SEQUENTIAL,INTERNAL,FIXED 80
145 DISPLAY AT(6,1)BEEP:"--> Codefeld wird eingelesen"
150 FOR I=32 TO 143
160 INPUT #1:GRAPH$(I-32),ASCII$
170 NEXT I
180 CLOSE #1
190 DISPLAY AT(8,1):"Name der Bildschirmdatei     (GERAET.NAME)"
200 ACCEPT AT(10,1)BEEP:DATEI$
210 ON ERROR 200
220 OPEN #1:DATEI$,INPUT ,SEQUENTIAL,INTERNAL,FIXED 80
230 DISPLAY AT(11,1)BEEP:"--> Copyfeld wird eingelesen"
240 FOR I=1 TO 24
250 INPUT #1:BILD$(I)
260 NEXT I
265 CLOSE #1
270 DISPLAY AT(13,1)BEEP:"--> Umwandlung laeuft            BITTE WARTEN"
280 ON ERROR STOP
290 FOR I=1 TO 24
300 COPY$(I)=""
310 FOR K=1 TO 30
320 COPY$(I)=COPY$(I)&GRAPH$(ASC(SEG$(BILD$(I),K,1))-32)
330 NEXT K
340 NEXT I
350 DISPLAY AT(16,1):"Hardcopy abspeichern (J/N)"
360 ACCEPT AT(16,28)SIZE(1)VALIDATE("JN")BEEP:SKP$
370 IF SKP$<>"J" THEN 470
380 DISPLAY AT(17,1):"Name der Hardcopydatei       (GERAET.NAME)"
390 ACCEPT AT(19,1)BEEP:DATEI$
400 IF SEG$(DATEI$,1,3)<>"DSK" THEN 1000
410 ON ERROR 390
420 OPEN #1:DATEI$,OUTPUT,SEQUENTIAL,INTERNAL,FIXED 255
425 ON ERROR STOP
430 FOR I=1 TO 24
440 PRINT #1:COPY$(I)
450 NEXT I
460 CLOSE #1
470 GOSUB 500
480 GOTO 580
500 FOR I=17 TO 19 :: DISPLAY AT(I,1):"" :: NEXT I
510 DISPLAY AT(17,1):"Hardcopy wird ausgedruckt"
520 OPEN #1:"PIO",VARIABLE 255
530 PRINT #1:GS$
540 FOR I=1 TO 24 :: PRINT #1:TAB$;G$;COPY$(I)
550 NEXT I
560 PRINT #1:N$
570 CLOSE #1
```

```
575 RETURN
580 DISPLAY AT(19,1):"Weitere Umwandlungen (J/N)"
590 ACCEPT AT(19,28)SIZE(1)VALIDATE("JN")BEEP:SKP$
600 IF SKP$="J" THEN CALL HCHAR(8,1,32,544):: GOTO 190
610 CALL CLEAR
620 STOP
1000 DISPLAY AT(17,1):"Eine Abspeicherung der      Hardcopy ist nur mit einem  D
iskettenlaufwerk moeglich !"
1010 CALL SOUND(250,110,0):: GOTO 350
2000 CALL HCHAR(3,1,32,704)
2005 DISPLAY AT(4,1):"Name der gespeicherten      Hardcopy (GERAET.NAME)"
2010 ACCEPT AT(6,1)BEEP:DATEI$
2020 IF SEG$(DATEI$,1,3)<>"DSK" THEN 3000
2025 ON ERROR 2010
2030 OPEN #1:DATEI$,INPUT ,SEQUENTIAL,INTERNAL,FIXED 255
2035 ON ERROR STOP
2040 FOR I=1 TO 24
2050 INPUT #1:COPY$(I)
2060 NEXT I
2065 CLOSE #1
2070 GOSUB 500
2080 DISPLAY AT(8,1):"Weitere Ausdrucke (J/N)"
2090 ACCEPT AT(8,25)VALIDATE("JN")SIZE(1)BEEP:SKP$
2100 IF SKP$="J" THEN 2000
2110 CALL CLEAR
2120 STOP
3000 DISPLAY AT(8,1):"Nur mit Diskette moeglich !"
3010 CALL SOUND(250,110,0)
3020 GOTO 2010
5000 DISPLAY AT(4,1):"1 - Umwandeln einer Hardcopy2 - Ausdruck  einer Hardcopy"
5010 ACCEPT AT(6,1)SIZE(1)VALIDATE("12")BEEP:M
5020 ON M GOTO 110,2000
5030 END
```



LASER 110 – 4 KByte RAM. Dieser „persönliche Computer" eignet sich als Helfer beim Studium, am Arbeitsplatz oder bei Analysen und Statistiken. Mit seinen Peripheriegeräten erfüllt er alle Ansprüche an einen Home-Computer.

Die Erde wird von einem Weltraummonster, dem bösen Göört bedroht.

Dieser böse Göört kann jedoch nicht ungesehen die Erde erreichen, da die Radaranlagen ihn sofort erkennen würden, und er keine Chance hätte, zur Erde vorzudringen.
Aus diesem Grund schickt er Wellen von Ufos zur Erde, um die Radaranlagen zu vernichten.
Wenn es den UFOs gelingt, alle fünf Radartürme zu zerstören, so kann der böse Göört zur Erde vordringen.
Es liegt nun an Ihnen, diese Invasion zu verhindern. Schießen Sie die UFOs mit Ihrem Abfangjäger SILVERSPAR ab, bevor diese sämtliche Radartürme vernichtet haben.
Sollte Ihnen dies nicht gelingen, so müssen Sie gegen den bösen Göört höchstpersönlich antreten und mit ihm kämpfen.

SILVERSPAR ist ein Spiel für den TI 99/4 A, welches in Extended Basic geschrieben wurde und mittels Joystick 1 gesteuert wird.

SILVERSPAR ist eine Herausforderung an Ihr Reaktionsvermögen.
Tippen Sie das Listing ein, doch bevor Sie das Programm mit „RUN" starten, schreiben Sie dieses auf Kassette oder Diskette weg.
Das Programm besitzt einen Copyrightschutz, der beim Weglassen oder Verändern der Copyrightzeilen das System „abstürzen" läßt. Der Rechner kann dann nur noch mit <FCTN><QUIT> oder ausschalten neu initialisiert werden.
Nach dem Programmstart mit „RUN" dauert es einige Sekunden, bis der Rechner die interne STACK-Liste erstellt hat. Jetzt wird der Bildschirm für ca. 30 Sekunden schwarz. Das Titelbild erscheint und die Titelmelodie wird gespielt.
Zum Spielstart drücken Sie den Aktionsknopf am Joystick 1. 10 Sekunden später beginnt das Spiel
Am unteren Bildschirmrand werden jetzt die Kirche, das Haus und ein Radarturm aufgebaut. Der SILVERSPAR erscheint am rechten Bildschirmrand.
Steuern Sie mit dem Joystick den SILVERSPAR nach rechts, links, oben und unten. Der SILVERSPAR bewegt sich in Spielszene 1 nur sehr langsam in horizontaler Richtung. Dabei bewegt sich die Stadt jedoch relativ schnell in entgegengesetzter Richtung, so daß der Eindruck eines schnellen Fluges entsteht.
Gleichzeitig mit dem SILVERSPAR wird ein UFO auf dem Bildschirm sichtbar, welches es gilt abzuschießen.
SILVERSPAR kann nur horizontal schießen. Um einen Schuß abzufeuern, muß der Aktionsknopf gedrückt und gleichzeitig der Joystick in die gewünschte Richtung bewegt werden (rechts oder links).
Die UFOs wechseln laufend die Farbe. Je nachdem, welche Farbe das UFO beim Abschuß hatte, erhalten Sie
UFO Typ 1: 10 bis 100 Punkte
UFO Typ 2: 20 bis 200 Punkte
UFO Typ 3: 30 bis 300 Punkte
UFO Typ 4: 40 bis 400 Punkte
Die UFOs versuchen, die Radartürme abzuschießen. Wird ein Radarturm getroffen, so wird dieser viermal durch einen neuen Turm ersetzt.
Nach 10 abgeschossenen UFOs ist ein leises, dann immer lauter werdendes „Schnarren" zu hören.
Dies ist das Zeichen, daß eine neue Welle UFOs naht.
Gelingt es Ihnen in einer Welle alle 10 UFOs abzuschießen, ohne daß hierbei ein Radarturm abgeschossen wurde, so erhalten Sie zu Ihren anfänglichen 5 SILVERSPARs einen zusätzlichen Abfangjäger hinzu.
Kollidiert SILVERSPAR jedoch mit einem UFO, so verliert er ein „Leben".
Wenn die UFOs alle 5 Radartürme zerstört haben, so erlischt die Stadt und eine Sirene wird lautstark hörbar. Der böse Göört kommt jetzt ins Bild.
Um eine neue Stadt mit weiteren 5 Radartürmen zu bekommen, müssen Sie den bösen Göört besiegen. Es genügt jedoch nicht, Göört mit einem einzigen Treffer zu erledigen. Göört muß zehnmal getroffen werden, bevor er vernichtet ist.
Mit jedem Treffer wechselt der böse Göört die Farbe und Sie erhalten, je nach Farbe, 50 bis 500 Punkte. Wenn Sie Göört zehnmal getroffen haben, so explodiert er und Sie erhalten eine neue Stadt.
Von Zeit zu Zeit schleudert Göört einen Feuerwirbel gegen den SILVERSPAR. Diesem Wirbel gilt es auszuweichen, denn eine Berührung mit dem Feuerwirbel oder mit dem bösen Göört kostet ein „Leben".
Das Spiel ist dann zu Ende, wenn alle SILVERSPARs zerstört sind.

Doch jetzt genug der Erklärungen. Ich wünsche ein ERRORfreies Eintippen und viel Spaß beim Spiel. Nachfolgend noch einige Tips zum Eintippen:

1) Einige Zeilen sind länger, als es der TI zuläßt. Tippen Sie die Zeile ein und schicken Sie diese mit <ENTER> ab. Danach holen Sie die Zeile mit <FCTN><REDO> wieder zurück und können den Rest eingeben.
2) Benutzer eines Diskettenlaufwerkes ohne Speichererweiterung sollten das Spiel bei abgeschalteter Peripheriebox eintippen und über einen Kassettenrecorder abspeichern, da durch den Dateibuffer des Laufwerkes der Speicher beim Programmlauf zu klein ist.
3) Wenn Sie ein Laufwerk mit Erweiterung besitzen, so ist vor dem Einlesen des Programmes von Kassette die Befehlsfolge: CALL FILES(1) <ENTER> NEW <ENTER> einzugeben, damit das Programm von Kassette eingelesen werden kann.
4) Während des Spielablaufes darf die <ALPHA LOCK>-Taste nicht eingerastet sein, da sonst der Joystick nicht richtig abgefragt wird.
5) Tippen Sie vor allem die DATA-Zeilen sehr genau ab, da ein hieraus resultierender Fehler sehr schwer zu finden ist.

```
100 REM ********************************
110 REM *                              *
120 REM *      S I L V E R S P A R     *
130 REM *------------------------------*
140 REM * ein Spiel fuer den TI 99/4 A *
150 REM * geschrieben in Extended BASIC *
160 REM * Steuerung mittels Joystick 1 *
170 REM *------------------------------*
180 REM * (c) 1984 by Volker Becker    *
190 REM *                              *
200 REM ********************************
210 REM
220 REM
230 ON ERROR 1600
240 CPR$="@ 1984 VOLKER BECKER"
250 SKP=ASC(SEG$(CPR$,8,1))-ASC(SEG$(CPR$,20,1))-4
260 IF SKP THEN 1600 ELSE 920
270 RANDOMIZE
280 ON ERROR STOP
290 DATA 104,112,120,124,136,140
300 OPTION BASE 1
310 DIM C(6),CU(4,2),B(6)
320 RESTORE 290
330 FOR I=1 TO 6 :: READ C(I):: NEXT I
```

```
340 CALL CLEAR :: CALL SCREEN(2):: CALL GRAPH
350 RESTORE 510
360 FOR I=1 TO 10 :: READ COL(I):: NEXT I
370 CALL HCHAR(2,1,30,704)
380 CALL CHARSET :: CALL SCREEN(2)
390 FOR I=1 TO 100 :: CALL HCHAR(RND*21+2,RND*30+1,91):: NEXT I
400 DATA 96,100,64,68,72,76,80,84
410 RESTORE 400
420 FOR I=1 TO 4 :: FOR K=1 TO 2 :: READ CU(I,K):: NEXT K :: NEXT I
430 CALL TITEL(COL(),B(),CPR$,STV):: P,UN=0 :: SPAR=4
440 CALL ANZSPAR(SPAR):: CALL PUNKTE(P)
450 CALL MINICHAR
460 CALL COLOR(1,2,6,3,2,6,4,2,6,8,11,1)
470 CALL SPRITE(£5,132,6,185,25,£4,104,6,185,110,£6,129,6,185,195)
480 FOR I=4 TO 6 :: FOR K=184 TO 169 STEP -1 :: CALL POSITION(£I,Z,S):: CALL LOC
ATE(£I,K,S):: CALL SOUND(-100,I*100-K,0,I*100-K+2,0,I*100-K-2,0):: NEXT K :: NEX
T I
490 CO=1
500 TRSS,STV,TRUFO=0
510 DATA 7,9,10,11,12,15,16,8,4,14
520 UN=UN+1 :: IF UN>4 THEN UN=1
530 FOR I=4 TO 6 :: CALL MOTION(£I,0,0):: NEXT I :: XA=0
540 CALL SPRITE(£1,108,11,89,30,0,0)
550 CALL SPRITE(£2,44,15,RND*120+20,RND*170+50):: CALL COLOR(£2,16):: FOR K=1 TO
 5 :: CALL SOUND(-20,-7,10-2*K):: NEXT K :: CALL COLOR(£2,15)
560 CALL PATTERN(£2,CU(UN,1)):: CALL COLOR(£2,COL(CO))
570 CALL COLOR(£4,3)
580 CALL POSITION(£2,VZ2,VS2):: IF VZ2<60 THEN SG=1 ELSE IF VZ2>150 THEN SG=-1 E
LSE SG=SGN(RND*1.5-1)
590 CALL MOTION(£2,RND*2*SG,RND*10*(SGN(RND*1.5-1))):: CALL COLOR(8,COL(RND*9+1)
,1)
600 FOR LF=1 TO 10
610 CO=CO+1 :: IF CO>10 THEN CO=1
620 CALL COLOR(£2,COL(CO))
630 FOR I=1 TO 2 :: CALL JOYST(1,X,Y):: CALL PATTERN(£4,C(I),£2,CU(UN,I))
640 CALL POSITION(£1,VZ1,VS1,£2,VZ2,VS2,£4,VZ3,VS3):: IF ABS(VS2-VS3)<12 THEN 76
0
650 IF X=XA THEN 670
660 FOR K=4 TO 6 :: CALL MOTION(£K,0,-X*5):: NEXT K
670 IF X=0 THEN 690
680 IF X=-4 THEN CALL PATTERN(£1,116)ELSE CALL PATTERN(£1,108)
690 CALL MOTION(£1,-Y*3,X)
700 CALL COINC(£1,£2,12+RUNDE,ST):: IF ST THEN 1280
710 CALL KEY(1,T,S):: IF T=18 THEN 1140
720 XA=X
730 NEXT I
740 NEXT LF
750 GOTO 580
760 CALL SPRITE(£3,60,16,VZ2+8,VS2,127,0):: CALL SOUND(-50,-3,0):: CALL POSITION
(£4,VZ3,VS3):: CALL DELSPRITE(£3):: IF ABS(VS2-VS3)<15 THEN Q=4 :: GOSUB 780 ::
GOTO 840
770 GOTO 650
780 CALL MOTION(£1,0,0)
790 CALL PATTERN(£Q,36):: CALL COLOR(£Q,12):: CALL SOUND(-100,-7,0)
800 CALL PATTERN(£Q,40):: CALL SOUND(-100,-6,0)
810 CALL PATTERN(£Q,44):: CALL SOUND(-100,-5,0)
820 CALL COLOR(£Q,16)
830 RETURN
840 CALL POSITION(£4,VZ3,VS3)
850 CALL DELSPRITE(£4)
860 CALL MOTION(£1,0,0,£2,0,0,£4,0,0,£5,0,0,£6,0,0):: XA=0
870 TRUFO=TRUFO+1 :: STV=-1 :: IF TRUFO=5 THEN 950
```

```
880 CALL SPRITE(£4,104,6,185,VS3)
890 TRO=200+100*TRUFO
900 FOR K=184 TO 169 STEP -1 :: CALL POSITION(£4,Z,S):: CALL LOCATE(£4,K,S):: CA
LL SOUND(-100,TRO-K,0,TRO-K+2,0,TRO-K-2,0):: NEXT K
910 GOTO 520
920 RANDOMIZE ASC(SEG$(CPR$,9,1))
930 FOR I=8 TO 20 :: SUM=SUM+INT(RND*ASC(SEG$(CPR$,I,1))):: NEXT I
940 GOTO 1460
950 CALL DELSPRITE(ALL):: CALL SPRITE(£2,120,14,247,120,2,0):: CALL SIRENE
960 CALL MOTION(£2,0,0)
970 C1=1 :: CALL COLOR(£2,COL(C1))
980 CALL SPRITE(£1,108,11,89,30,0,0)
990 CALL LOCATE(£2,89,120)
1000 CALL POSITION(£2,VZ2,VS2):: IF VZ2<60 THEN SG=1 ELSE IF VZ2>150 THEN SG=-1
ELSE SG=SGN(RND*1.5-1)
1010 CALL MOTION(£2,RND*2*SG,RND*10*(SGN(RND*1.5-1))):: CALL COLOR(8,COL(RND*9+1
),1)
1020 FOR LF=1 TO 11-RUNDE
1030 CO=CO+1 :: IF CO>10 THEN CO=1
1040 FOR I=3 TO 4
1050 CALL JOYST(1,X,Y):: CALL POSITION(£1,VZ1,VS1,£2,VZ2,VS2):: CALL PATTERN(£2,
C(I))
1060 IF X=0 THEN 1080
1070 IF X=-4 THEN CALL PATTERN(£1,116)ELSE CALL PATTERN(£1,108)
1080 CALL MOTION(£1,-Y*3,X*3)
1090 CALL COINC(£1,£2,12+RUNDE,ST):: IF ST THEN 1430
1100 CALL KEY(1,T,S):: IF T=18 THEN 1310
1110 NEXT I
1120 NEXT LF
1130 GOTO 1490
1140 IF X=0 THEN 720
1150 CALL MOTION(£1,0,0)
1160 CALL POSITION(£1,VZ1,VS1,£2,VZ2,VS2)
1170 IF (VS1+2*X)<1 OR(VS1+2*X)>255 THEN 720
1180 CALL SPRITE(£3,92,16,VZ1,VS1+2*X,0,31*X):: CALL SOUND(-50,-2,0):: IF SGN(VS
2-VS1)<>SGN(X)THEN CALL DELSPRITE(£3):: GOTO 720
1190 CALL DELSPRITE(£3):: IF ABS(VZ1-VZ2)<12 THEN Q=2 :: GOSUB 780 :: GOTO 1210
1200 GOTO 720
1210 P=P+10*CO*UN :: CALL PUNKTE(P):: TRSS=TRSS+1 :: IF TRSS=10 THEN 1230
1220 GOTO 520
1230 TRSS=0 :: RUNDE=RUNDE+1 :: IF STV THEN 1260 ELSE SPAR=SPAR+1
1240 IF SPAR>15 THEN SPAR=15
1250 CALL ANZSPAR(SPAR):: CALL BIM
1260 IF RUNDE>8 THEN RUNDE=8
1270 CALL NEXTR :: STV=0 :: GOTO 520
1280 CALL DELSPRITE(£2):: Q=1 :: GOSUB 780
1290 CALL DELSPRITE(£1):: CALL COLOR(£1,1):: SPAR=SPAR-1 :: CALL ANZSPAR(SPAR)::
 IF SPAR<0 THEN 1480
1300 GOTO 520
1310 IF X=0 THEN 1110
1320 CALL MOTION(£1,0,0)
1330 CALL POSITION(£1,VZ1,VS1,£2,VZ2,VS2)
1340 IF (VS1+2*X)<1 OR(VS1+2*X)>255 THEN 1110
1350 CALL SPRITE(£3,92,16,VZ1,VS1+2*X,0,31*X):: CALL SOUND(-50,-2,0):: IF SGN(VS
2-VS1)<>SGN(X)THEN CALL DELSPRITE(£3):: GOTO 1110
1360 CALL DELSPRITE(£3):: IF ABS(VZ1-VZ2)<12 THEN 1380
1370 GOTO 1110
1380 P=P+50*C1 :: CALL PUNKTE(P):: C1=C1+1 :: IF C1>10 THEN 1410
1390 CALL SOUND(-100,-7,0):: CALL SOUND(-100,-6,0):: CALL SOUND(-100,-5,0)
1400 CALL COLOR(£2,COL(C1)):: GOTO 1110
1410 Q=2 :: GOSUB 780 :: CALL DELSPRITE(ALL):: RUNDE=RUNDE+1 :: IF RUNDE>8 THEN
RUNDE=8
```

```
1420 GOTO 470
1430 CALL DELSPRITE(£3):: Q=1 :: GOSUB 780
1440 CALL COLOR(£1,1):: SPAR=SPAR-1 :: CALL ANZSPAR(SPAR):: IF SPAR<0 THEN 1480
1450 GOTO 980
1460 SKP=INT(RND*VAL(SEG$(CPR$,3,4)))-184-2*SUM
1470 IF SKP THEN 1600 ELSE 270
1480 CALL DELSPRITE(ALL):: STV=-1 :: CALL CHARSET :: GOTO 430
1490 CALL MOTION(£1,0,0,£2,0,0)
1500 CALL POSITION(£1,VZ1,VS1,£2,VZ2,VS2):: CALL SPRITE(£3,136,10,VZ2,VS2,(VZ1-V
Z2)/2,(VS1-VS2)/2)
1510 FOR I=1 TO 2
1520 FOR K=1 TO 2
1530 CALL SOUND(-100,-7,I*K)
1540 CALL PATTERN(£2,C(K+2),£3,C(K+4))
1550 CALL JOYST(1,X,Y):: CALL MOTION(£1,-Y*6,X*6)
1560 CALL COINC(£1,£2,11+R,ST1):: CALL COINC(£1,£3,18,ST2):: IF ST1 OR ST2 THEN
1430
1570 NEXT K
1580 NEXT I
1590 CALL DELSPRITE(£3):: CALL MOTION(£1,0,0):: GOTO 1000
1600 ON BREAK NEXT
1610 CALL SCREEN(16):: CALL CLEAR
1620 CALL SOUND(-1000,1000,0,998,0,1002,0):: GOTO 1620
1630 SUB BIM
1640 CALL MOTION(£1,0,0,£4,0,0,£5,0,0,£6,0,0):: CALL DELSPRITE(£2)
1650 FOR I=0 TO 30 :: CALL SOUND(-100,523,I,525,I,521,I):: NEXT I
1660 SUBEND
1670 SUB NEXTR
1680 CALL MOTION(£1,0,0,£4,0,0,£5,0,0,£6,0,0):: CALL DELSPRITE(£2)
1690 FOR I=30 TO 0 STEP -2
1700 CALL SOUND(-100,-7,I):: CALL SOUND(-100,-6,I):: CALL SOUND(-100,-5,I):: CAL
L SOUND(-100,-6,I):: NEXT I
1710 SUBEND
1720 SUB MINICHAR
1730 RESTORE 2240
1740 FOR I=32 TO 95 :: READ C$ :: CALL CHAR(I,C$):: NEXT I
1750 SUBEND
1760 SUB ANZSPAR(SPAR)
1770 CALL HCHAR(1,3,32,15)
1780 IF SPAR<=0 THEN SUBEXIT
1790 CALL HCHAR(1,3,59,SPAR)
1800 SUBEND
1810 SUB PUNKTE(P)
1820 DISPLAY AT(1,23)SIZE(6):USING "£££££":P
1830 SUBEND
1840 SUB TITEL(COL(),B(),CPR$,STV)
1850 DATA S,I,L,V,E,R,S,P,A,R
1860 DATA G,A,M,E,O,V,E,R
1870 IF STV THEN RESTORE 1860 ELSE RESTORE 1850
1880 CALL MAGNIFY(2)
1890 IF STV THEN 1910
1900 CALL HCHAR(24,1,32,64)
1910 CALL CHAR(91,"0000001818",64,"3C4299A1A199423C"):: FOR I=1 TO 7 :: CALL COL
OR(I,1,8):: NEXT I :: CALL COLOR(8,11,1)
1920 IF STV THEN 2030
1930 S=48 :: Z=17
1940 FOR I=1 TO 10 :: GOSUB 1990 :: READ C$ :: CALL SPRITE(£I,ASC(C$),COL(I),Z,S
):: Z=Z+16 :: S=S+16 :: NEXT I
1950 DISPLAY AT(24,5)SIZE(20):CPR$
1960 CALL FLYINGDREAM(B(),1)
1970 CALL DELSPRITE(ALL):: CALL MAGNIFY(3):: CALL HCHAR(24,1,32,64):: CALL COLOR
(3,8,8,4,8,8)
```

```
1980 GOTO 2110
1990 CALL SPRITE(£I,42,COL(I),Z,S):: CALL SOUND(-100,-7,0)
2000 CALL PATTERN(£I,43):: CALL SOUND(-100,-6,0)
2010 CALL PATTERN(£I,91):: CALL SOUND(-100,-5,0)
2020 RETURN
2030 ZP,SP=72 :: I=1
2040 FOR J=0 TO 1
2050 FOR K=0 TO 3
2060 Z=ZP+J*32 :: S=SP+K*32 :: READ C$ :: GOSUB 1990 :: CALL SPRITE(£I,ASC(C$),C
OL(I),Z,S)
2070 I=I+1
2080 NEXT K
2090 NEXT J
2100 GOTO 1950
2110 SUBEND
2120 SUB SIRENE
2130 K=0
2140 FOR I=261 TO 293 STEP 10 :: CALL SOUND(-250,I,0,I-2,0):: NEXT I :: CALL PAT
TERN(£2,124)
2150 K=K+1
2160 FOR I=303 TO 523 STEP 10 :: CALL SOUND(-250,I,0,I-2,0,I-42,3):: NEXT I :: C
ALL PATTERN(£2,120)
2170 FOR I=523 TO 491 STEP -10 :: CALL SOUND(-250,I,0,I-2,0,481-(523-I),3):: NEX
T I :: CALL PATTERN(£2,124)
2180 FOR I=481 TO 261 STEP -10 :: CALL SOUND(-250,I,0,I-2,0,I+42,3):: NEXT I ::
CALL PATTERN(£2,120)
2190 IF K=3 THEN 2220
2200 FOR I=261 TO 293 STEP 10 :: CALL SOUND(-250,I,0,I-2,0,293+(293-I),3):: NEXT
 I :: CALL PATTERN(£2,124)
2210 GOTO 2150
2220 SUBEND
2230 SUB GRAPH
2240 DATA ,,,,00000001E273351F,0B060F1B3E3F61C1,2870E0C08CD8F058
2250 DATA CCE6B3E0C08,00001800C233141F,0B000F110E3B01C1,287086400C18F058,CCE2B3E
0028C,1000100080220411,0A000510042A0041
2260 DATA 2000044004100005,048022800288,0018242424242418,0010301010101038,00182404081
03C,0018240804241 8,000818283C0808
2270 DATA 003C2038042418,0018203824241 8,003C040810101,00182418242418,0018242 41C0
418,,0070380E0E387
2280 DATA 0101010101010101,0101010101010101,8080808080808080,8080808080808080,0103
070F1E3F7FE3,E37F3F1F0C0C0C3C,80C0E0F078FCFEC7
2290 DATA C7FEFCF830303 03C,0000000018 3C7EE7,E37F3F1F0C0E,0000000018 3C7EE7,C7FEFC
F8307,2010080503 0F3FFF,C1FF090D0701
2300 DATA 040810A0C0F0FCFF,83FF90B0E08,0000003C030F3FFF,F3FF09080818,0000003CC0F
0FCFF,CFFF90101018,003C66C3FF7F3F01
2310 DATA 03060C060301,003C66C3FFFEFC8,C0603060C08,00007EC3FF1C3F01,03060606 0301
,00007EC3FF38FC8,C0606060C08
2320 DATA ,,,0000001818,0000000000000FF,FF,0000000000000FF,FF
2330 DATA 0000000103070 77F,FFCEFF7F,00000080C0E0E0FE,FF73FFFE,0000000000003077F,F
FF1FF7F
2340 DATA 0000000000C0E0FE,FF8FFFFE,0001030 70D0B0F07,0302030307 0D1F3B,0080C060B0
B070E,C0C040C0E070B8DC,00307FD8A05F2C17
2350 DATA 172C5FA0D87F3,0000800000F00CFF,FF0CF000008,0001030 60D0D0E07,0303020307
0E1D3B,0080C0E0B0D0F0E,C040C0C0E0B0F8DC
2360 DATA 0000010000 0F30FF,FF300F000001,000CFE1B05FA34E8,E834FA051BFE0C,83CFFF7F
61333E3D,1D1C1F0C07030301,C1F3FFFE86CC7CBC
2370 DATA B838F830E0C0C08,034FFF7F73213F3E,1D3C3F1E0F070301,C0F2FFFECE84FC7C,B83
CFC78F0E0C08,0000000000000003F,7FDBDBFFFFFFFFFF
2380 DATA 04040A0A1B1F1F95,D57F7FFFFFFFFFBFB,0000081C367FFF49,7F7F77777F7FFFFF,00
00000000000008,0000000080C0E0F,0001030606034143
2390 DATA 673D1800000003,00C000000018BCE6,C282C06060C08,00070C18180C0701,4342663
```

```
C18,000000183C6642C2,80E030181830E
2400 RESTORE 2330
2410 FOR I=96 TO 143 :: READ C$ :: CALL CHAR(I,C$):: NEXT I
2420 SUBEND
2430 SUB FLYINGDREAM(B( ),KY)
2440 DATA 1,195,261,329,391,523,659,1,261,329,1,293,349,4,329,391
2450 DATA 1,523,440,349,261,220,174,1,329,391,1,349,440,4,293,349
2460 DATA 1,195,246,349,391,493,698,1,293,349,1,261,329,2,246,293,2,293,349
2470 DATA 1,659,523,391,329,261,195,6,261,329
2480 DATA 2,30000,30000,30000,30000,30000,30000,1,30000,329,1,30000,369,2,30000,
293,2,30000,440,1,30000,440
2490 DATA 1,30000,493,4,30000,391,1,174,174,220,220,261,261,2,30000,30000,2,293,
349,2,349,440
2500 DATA 1,195,195,246,246,293,293,6,329,391,4,195,261,329,391,523,659
2510 DATA 1,195,329,1,174,293,2,164,261,2,329,391
2520 DATA 1,195,329,1,174,293,2,164,261,2,329,391
2530 DATA 1,195,329,1,174,293,2,164,261,2,329,391
2540 DATA 6,164,261,0
2550 RESTORE 2440
2560 AZ,STV=0
2570 READ K :: IF K=0 THEN 2550
2580 READ B(1),B(2),B(3),B(4),B(5),B(6)
2590 FOR I=1 TO K
2600 FOR J=1 TO 6
2610 STV=STV+1 :: IF STV>AZ THEN READ AZ,M1,M2 :: STV=1
2620 CALL SOUND(350,M1,1,M2,0,B(J),5):: CALL KEY(1,T,S):: IF S<>0 THEN SUBEXIT
2630 NEXT J :: NEXT I
2640 GOTO 2570
2650 SUBEND
```

TRANSFER

Das Programm TRANSFER gehört mit zu den ASCII Hilfsprogrammen und erweitert dieses Programmpaket.

Das Unterprogramm ASCIISET belegt den Zeichensatz durch das mittels des Programmes ASCIIDEF erstellte Codefeld und erfordert immer einen externen Speicher. Mittels des Programmes TRANSFER wird aber vom Rechner selbst ein Unterprogramm geschrieben, welches den Zeichensatz im Programm, also ohne externen Speicher, umbelegt.
TRANSFER ist also ein Programm, welches aus dem Codefeld ein Unterprogramm schreibt.
Dieses Unterprogramm kann dann mittels MERGE auf das gewünschte BASIC-Hauptprogramm aufgeladen werden.
Das Programm TRANSFER ist nur mittels eines Diskettenlaufwerkes sinnvoll einzusetzen, da man von einem Kassettenrecorder kein Programm mit MERGE aufladen kann.
Nach dem Programmstart wird man nach dem entspr. Codefeld gefragt, welches mit Gerät.-Name (z. B. DSK1.FELD) eingegeben werden muß.
Danach wird man nach dem Namen gefragt, unter welchem das Unterprogramm auf Diskette gespeichert werden soll (z. B. DSK1.PGM).
Zuletzt kommt die Frage nach dem Namen des zu erstellenden Unterprogrammes (z. B. ASCII).
Der Rechner generiert jetzt das Unterprogramm, in folgender Form:
Das Unterprogramm belegt die Zeilen 32000 bis 32019. In der ersten Zeile steht SUB und der eingegebene Name
Zeilen 32001 bis 32016 sind DATA-Zeilen, in welchen der komplette hexadezimale Zeichensatz abgelegt ist.
Zeile 32017 ist die RESTORE Anweisung.
Zeile 32018 enthält die Schleife zur Umbelegung des Zeichensatzes.
In Zeile 32019 steht SUBEND.
Das so generierte Unterprogramm kann nun mit MERGE auf ein bestehendes BASIC-Programm aufgeladen werden und wird im Programm mit CALL Name (z. B. CALL ASCII) aufgerufen und belegt den kompletten Zeichensatz um, wie er mittels „ASCIIDEF" generiert wurde.
Der TI-99/4 A programmiert sich somit praktisch selbst.

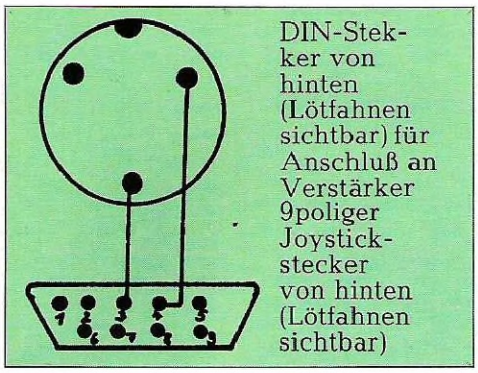
```
100 REM ******************************
110 REM *                            *
120 REM *      T R A N S F E R       *
130 REM *----------------------------*
140 REM *   TI 99/4 A                *
150 REM * + TI Extended BASIC        *
160 REM * + Diskettenlaufwerk        *
170 REM *----------------------------*
180 REM * (c) 1984 by Volker Becker *
190 REM *                            *
200 REM ******************************
```

```
210 ON WARNING NEXT
220 DISPLAY AT(1,1)ERASE ALL:"Erstellen eines BASIC Unter- Programmes zur Umbele
gung       des Zeichensatzes       ------------------------------"
230 ON ERROR 710
240 DISPLAY AT(6,1):"Name der Eingabedatei       (Codefeld) :" :: ACCEPT AT(7,14
)BEEP:IDAT$
250 OPEN #1:IDAT$,INPUT ,SEQUENTIAL,INTERNAL,FIXED 80
260 ON ERROR 720
270 DISPLAY AT(9,1):"Ausgabedatei (Name unter    welchem das Programm auf    Dis
kette gespeichert werden soll)"
280 ACCEPT AT(12,7)BEEP:ODAT$
290 OPEN #2:ODAT$,OUTPUT,SEQUENTIAL,DISPLAY ,VARIABLE 163
300 ON ERROR STOP
310 DISPLAY AT(14,1):"Unterprogrammname :" :: ACCEPT AT(15,1)BEEP:NAME$
320 DISPLAY AT(18,1):"Unterprogramm ";NAME$ :: DISPLAY AT(19,1):"wird erstellt.
Bitte warten"
330 PRINT #2:CHR$(125)&CHR$(0)&CHR$(161)&CHR$(200)&CHR$(LEN(NAME$))&NAME$&CHR$(0
)
340 FOR I=1 TO 16
350 A$=CHR$(125)&CHR$(I)&CHR$(147)
360 FOR K=1 TO 7
370 INPUT #1:G$,HEX$
380 FOR J=16 TO 1 STEP -1
390 IF SEG$(HEX$,J,1)<>"0" THEN 420
400 NEXT J
410 GOTO 480
420 HEX$=SEG$(HEX$,1,J)
430 A$=A$&CHR$(200)&CHR$(LEN(HEX$))
440 FOR J=1 TO LEN(HEX$)
450 A$=A$&SEG$(HEX$,J,1)
460 NEXT J
470 IF K=7 THEN 490
480 A$=A$&CHR$(179)
490 NEXT K
500 A$=A$&CHR$(0)
510 PRINT #2:A$
520 NEXT I
530 DATA 125,17,148,201,125,1,0
540 DATA 125,18,140,73,190,200,2,51,50,177,200,3,49,52,51,130,151,67,36,130,157,
200,4,67,72,65,82
550 DATA 183,73,179,67,36,182,130,150,73,0
560 DATA 125,19,168,0
570 RESTORE 530
580 FOR I=1 TO 3
590 A$=""
600 READ WERT
610 A$=A$&CHR$(WERT)
620 IF WERT=0 THEN 630 ELSE 600
630 PRINT #2:A$
640 NEXT I
650 PRINT #2:CHR$(255)&CHR$(255)
660 CLOSE #1 :: CLOSE #2
670 DISPLAY AT(18,1):"Unterprogramm ";NAME$ :: DISPLAY AT(19,1):"ist erstellt un
d kann mit   MERGE ";ODAT$
680 DISPLAY AT(21,1)BEEP:"auf ein BASIC Programm      aufgeladen werden."
690 STOP
700 END
710 CLOSE #1 :: GOTO 240
720 CLOSE #2 :: GOTO 280
```

Konzertreife Stimme für den TI 99/4A

Wer den Sound seines TI 99 aufmotzen will, braucht dazu nur einen Radiorecorder und die nachfolgende Bauanleitung.

Wer sich mit dem flachen, dumpfen Ton eines 198 DM-Fernsehers herumschlagen muß, ist sicher dankbar für eine kleine Verbesserung der Tonmöglichkeiten.
Daß kleine Qualitätssteigerungen nicht unbedingt teuer und aufwendig sein müssen, beweist diese Bauanleitung.
Die Voraussetzungen dafür sind denkbar gut: An der Interfacebuchse für das Kassettenrecorderkabel liegt ein Audiosignal an, das sich zum Ansteuern eines Radiorecorders, eines einzelnen Verstärkers oder sogar einer Stereoanlage eignet. Dazu benötigt man lediglich einen 9poligen Joystickstecker, mit dem man den Tonausgang des Computers anzapft und einen passenden Stecker für den Anschluß an den Verstärker. Hier findet ein dreipoliger DIN-Stecker bei den meisten Radiogeräten Verwendung. Bei fast allen Radiorecordern reicht nach meiner Erfahrung eine einfache Verbindung zwischen Computer und Audioeingangsbuchse des betreffenden Gerätes aus. Will man einen HiFi-Verstärker oder ein ähnliches großes Gerät anschließen, ist jedoch fast immer noch ein Vorverstärker nötig, der zwischen Computer und Stereoanlage geschaltet wird.



DIN-Stekker von hinten (Lötfahnen sichtbar) für Anschluß an Verstärker 9poliger Joystickstecker von hinten (Lötfahnen sichtbar)

Die genaue Verdrahtung können Sie der Abbildung entnehmen. Der Tonausgang des TI 99 kann an der 9poligen Buchse an der Geräterückwand mit Pin 4 gegen Masse abgegriffen werden. Die Verbindung mit dem externen Verstärker stellt ein zweiadriges, nicht unbedingt abgeschirmtes NF-Kabel dar. Bei einem längeren Kabel empfiehlt sich trotzdem eine Abschirmung, um einen Netzbrummen zu vermeiden.
Nach der Verdrahtung sollte man unbedingt noch einmal alles auf seine Richtigkeit überprüfen, da Fehler sowohl Computer als auch den angeschlossenen Verstärker beschädigen könnten.
Will man den Tonausgang des TI 99 für die Ansteuerung eines externen Verstärkers benutzen, muß man die Recorderkabelbuchse jedesmal dafür freimachen. Deshalb für die User noch einen Tip, die sich auch das Recorderkabel selbst zusammengelötet haben: Der Verstärkeranschluß kann natürlich auch mit in dieses Kabel eingelötet werden. Dies ist bei gekauften Kabeln leider nicht möglich, da die Stekker fest verschweißt sind.
Noch viel Spaß beim Zusammenbau und bei den zukünftigen Videospielen, bei denen es ab jetzt kristallklar und hautnah kracht!

Stückliste: 1 Joystickstecker
1 Stecker, passend für benutzten Verstärker, evt. Vorverstärker für Stereoanlagen

Impressum: Computronic erscheint monatlich im Verlag: Tronic-Verlagsgesellschaft mbH, Landstr. 29, 3444 Wehretal 1, Tel.: (0 56 51) 4 06 43 / 4 06 93. Redaktion: (Verantw.) Rainer Beck, Axel Gredè, Frank Brall. Freie Mitarbeiter: G. Wagner, Volker Bekker, Rolf Freitag. Gesamtherstellung: D + V Paul Dierichs GmbH + Co. KG. Kassel. Vertrieb: Inland (Groß-, Einzel- und Bahnhofsbuchhandel), sowie Österreich und Schweiz: Verlagsunion, 6200 Wiesbaden. Anzeigenverwaltung: E. Herwig. Anzeigenpreis: Es gilt die Anzeigenliste Nr. 1. Bitte Media-Unterlagen anfordern. Bezugspreise: Einzelheft (Inland) DM 5,50, Abonnement (Inland) DM 55,-, (Ausland) DM 75,-. Autoren und Manuskripte: Bei Zusendung von Manuskripten und Datenträger erteilt der Autor dem Verlag die Genehmigung zum Abdruck und Versand der veröffentlichten Programme auf Datenträger. Für die mit Namen des Verfassers gekennzeichneten Beiträge übernimmt die Redaktion lediglich die presserechtliche Verantwortung. Alle in dieser Zeitschrift veröffentlichten Beiträge sind urheberrechtlich geschützt. Nachdruck (auch auszugsweise) und Vervielfältigung nur mit ausdrücklicher Genehmigung des Verlages. Für unaufgeforderte Einsendungen von Manuskripten, Tonträgern und Software wird keine Haftung übernommen.

Computronic Abonnement

Bitte ausschneiden und senden an:
Tronic-Verlag, Postfach 41, 3444 Wehretal 1.

Hiermit bestelle ich die Zeitschrift »Computronic« ab Heft Nr. _____ zum Jahresabonnementpreis (12 Ausgaben) von DM 55,- incl. Versand, Inland und DM 75,- incl. Versand, Ausland.

Name/Vorname:

Straße, Nr. Plz, Ort:

Ich wünsche folgende Zahlungsweise

☐ Bargeldlos durch Bankeinzug: Bankleitzahl:

Geldinstitut: Konto Nr.:

☐ gegen Rechnung

☐ gegen Vorkasse
(betreffendes bitte ankreuzen)

Datum, Unterschrift

Abonnements-Kündigungen:
6 Wochen vor Ablauf des Jahres-Abonnements.
Außerdem kann ich diese Bestellung innerhalb von 10 Tagen beim Verlag widerrufen!

Datum, Unterschrift

Ein Maskengenerator für den TI 99/4 A in Extended BASIC geschrieben

DISPLAY AT (Zeile, Spalte), SIZE (Länge), BEEP: USING „##.##", ACCEPT AT (Zeile, Spalte), SIZE (Länge), VALIDATE (Typ), BEEP: Welcher TI 99/4 A-Besitzer kennt nicht diese mühselige Prozedur der Maskenprogrammierung?

Solange die Eingabefelder immer in der gleichen Spalte liegen, und die Variablen alle vom gleichen Typ sind, kann man sich die Arbeit mit einer Schleife noch erleichtern. Werden jedoch die Felder an verschiedenen Positionen gebraucht, und sind die Variablen auch noch von verschiedenen Typen, so geht die große Zählerei auf dem Bildschirm los. Welcher TI-Besitzer hat sich in solchen Fällen nicht schon einmal ein Programm gewünscht, welches ihm diese mühselige und stupide Arbeit abnimmt? Bitte schön: Hier ist Ihr gewünschtes Programm oder besser gesagt das Programmpaket: SCREEN DESIGNER.

Das Programm wurde auf dem TI 99/4 A in TI Extended BASIC geschrieben, und erfordert einen Kassettenrecorder, oder ein Diskettenlaufwerk, um die Maske als FILE zu speichern.

Das Programm besteht aus drei Teilen:

Teil 1: SCREEN DESIGNER

Dieses Programm ist das Kernstück des Paketes. Es beinhaltet den Maskengenerator, sowie den Editor zum ändern einer bereits erstellten Maske.

Teil 2: SCREEN READER

Hierbei handelt es sich um das Unterprogramm MASKE, welches die mittels SCREEN DESIGNER erstellte Maske später in Ihrem BASIC-Programm verwaltet.

Teil 3: SCREEN PRINTER

Für dieses Programm brauchen Sie außer der oben schon genannten Peripherie zusätzlich noch einen Drucker, denn dieses Programm erstellt Ihnen eine Aufschlüsselung einer mit SCREEN DESIGNER erstellten Maske, und gibt diese auf dem Drucker aus. So haben Sie die entsprechende Maske immer griffbereit bei Ihren Programmunterlagen, und können jederzeit den Aufbau und die Gliederung der Maske vergleichen.

```
100 !@P-
110 REM ********************
120 REM * SCREEN  DESIGNER *
130 REM *------------------*
140 REM *   TI 99 / 4 A    *
150 REM * + Extended BASIC *
160 REM * + Ext. Speicher  *
170 REM *------------------*
180 REM *(c)1984 Tronicsoft*
190 REM *        V.Becker  *
200 REM ********************
210 REM
220 REM
230 !@P+
```

Um die Verwendung des Programmes SCREEN DESIGNER möglichst vielen TI-Benutzern zugänglich zu machen, wurde hier die Form eines FILES zur Speicherung der MASKE gewählt. Es wäre ohne weiteres möglich gewesen, die Maske auch als Programm erstellen zu lassen, dies setzt jedoch ein Diskettenlaufwerk voraus, um das Programm im MERGE-Modus erstellen zu lassen. Da die meisten TI-User jedoch kein Laufwerk zur Verfügung haben, ist der hier beschriebene Lösungsweg sicherlich am besten.

Doch nun zum eigentlichen Programm:

Wir beginnen eine Maske zu erstellen, indem wir das Programm SCREEN DESIGNER einladen (z. B. RUN „CS" oder RUN „DSK1.S-DESIGNER"). Das Programm startet nun von selbst, und das Titelbild erscheint. Sie haben jetzt zwischen zwei Möglichkeiten die Auswahl: 1. Sie wollen eine neue Maske erstellen oder 2. Sie wollen eine bereits erstellte Maske, die bereits auf Cassette oder Diskette gespeichert ist, ändern.

Bei Auswahl 2 werden Sie nun nach dem Namen der zu ändernden Maske gefragt. Geben Sie den Namen ein (z. B. CS1 oder DSK1.MASKE) und drücken Sie <ENTER>. Der Rechner lädt nun die gewünschte Maske ein, und gibt diese auf dem Leerbildschirm aus. Wenn Sie die Auswahl 1 getroffen haben, so geht der Rechner sofort in den Leerbildschirm, ohne etwas auszugeben.

Der Leerbildschirm hat eine gelbe Hintergrundfarbe und einen blauen Rand. In der Ecke ist ein L sichtbar. Bewegen Sie den in der oberen linken Ecke sichtbaren Cursor mittels den Cursor-Steuertasten über den Bildschirm. Schreiben Sie mittels der Tastatur Ihren Text an die gewünschten Stellen auf dem Bildschirm, oder überschreiben Sie den entsprechenden Text. Sie können an den Stellen, wo später die Eingabefelder liegen sollen, beliebige Zahlen oder Buchstaben eingeben. Vergessen Sie jedoch nicht die Dezimalstelle mit einem Punkt zu versehen, da der Rechner sonst bei der Generierung des USING-Formates keine Dezimalstellen generiert. Eine Betätigung der <ENTER>-Taste setzt den Cursor an die linke obere Bildschirmecke zurück. Nachdem Sie sich so Ihre Maske definiert haben, drücken Sie die Funktionstaste AID (FCTN 7), um den Leerbildschirm zu verlassen. Der Rechner speichert jetzt die Maske ab. Dieser Vorgang dauert ca. 50 sec. Zum Zeichen, daß das System arbeitet, wird der Rand des Bildschirmes rot.

Sobald die Maske abgespeichert ist, gelangen Sie in den Attributbildschirm. Der Rand hat jetzt die Farbe Weiß, und in der linken oberen Ecke ist ein A zu sehen.

Im Attributbildschirm wird die Lage der Ein- bzw. Ausgabefelder festgelegt. Wenn Sie eine bereits gespeicherte Maske editieren, so werden die Attribute der ursprünglichen Maske automatisch gesetzt. Bei einer Neudefinition erscheinen keine Attribute auf dem Bildschirm.

Die Attribute sehen wie folgt aus:

< Beginn eines Feldes

> Ende eines Feldes

Bewegen Sie den Cursor mit den Steuertasten an die gewünschte Stelle, und geben Sie dann das Attribut ein. Es können mit den Attributen auch Textstellen überschrieben werden. Dies macht der Maske nichts aus, da der Text bereits gespeichert ist, und durch ein Überschreiben nicht beeinträchtigt wird. Das entsprechende Feld wird immer zwischen Feldanfang- und Feldende-Attribut generiert. Die maximale Feldlänge darf jedoch 14 Zeichen nicht überschreiten. Ein Attribut kann durch die Funktionstaste ERASE (FCTN 3) gelöscht werden.

Es können maximal 25 Felder in einer Maske generiert werden.

Nachdem alle Attribute gesetzt wurden, verlassen Sie den Attributbildschirm wieder mit der Funktionstaste AID (FCTN 7). Jetzt führt der Rechner eine Fehlerkontrolle der Attribute durch.

Sollte ein Fehler aufgetreten sein, so meldet sich der Rechner mit einem akustischen Signal, und in der oberen linken Ecke erscheint ein F. Der Cursor hat jetzt die Farbe Weiß und steht an der fehlerhaften Bildschirmstelle.

```
240 DIM MASKE$(24),VALD$(25),FELD(25),FA(25),FE(25),USER$(25),TON(25)
250 GOTO 280 :: B$ :: DATEI$ :: VA$ :: X$ :: DEL :: I :: K :: MASKE :: S :: SA :
: SIGN :: SIGNALT :: SKIP :: SUCH :: T :: W1 :: W2 :: WA :: WE :: WERT :: X :: X
1 :: XS :: Y :: Y1 :: YS :: ZA
260 CALL CHAR :: CALL CLEAR :: CALL COLOR :: CALL DELSPRITE :: CALL GCHAR :: CAL
L HCHAR :: CALL KEY :: CALL MAGNIFY :: CALL SCREEN :: CALL SOUND :: CALL SPRITE
270 CALL VCHAR :: !@P-
280 FA(0),FE(0)=1
290 X=3 :: Y=1
300 GOTO 2070
310 CALL CLEAR :: CALL SCREEN(5):: CALL COLOR(1,2,5)
320 CALL CHAR(128,RPT$("0",16))
330 CALL CHAR(96,"FF818181818181FF")
340 FOR I=1 TO 2 :: CALL VCHAR(1,I,128,24):: CALL VCHAR(1,I+30,128,24):: NEXT I
350 CALL COLOR(1,2,11,2,2,11,3,2,11,4,2,11,5,2,11,6,2,11,7,2,11,8,2,11,9,2,11)
360 CALL COLOR(10,2,11,11,2,11,12,2,11)
370 ON SKIP GOTO 400,380
380 GOSUB 1430
390 FOR I=0 TO 24 :: MASKE$(I)="" :: NEXT I
400 CALL SPRITE(£2,76,16,8,8)
410 T=13 :: GOSUB 500
420 CALL KEY(0,T,S):: IF S=0 THEN 420
430 IF T=13 THEN GOSUB 500 ELSE 450
440 GOTO 420
450 IF T>7 AND T<12 THEN GOSUB 500 ELSE 470
460 GOTO 420
470 IF T=1 THEN 650
480 XS=X :: YS=Y :: SIGN=T :: T=9 :: GOSUB 500
490 CALL HCHAR(YS,XS,SIGN):: GOTO 420
500 X1=X :: Y1=Y
510 IF T=13 THEN X=3 :: Y=1 :: GOTO 530
520 ON T-7 GOSUB 580,550,610,630
530 CALL SPRITE(£1,96,2,Y*8-7,X*8-7)
540 RETURN
550 X=X+1 :: IF X>30 THEN X=3 :: Y=Y+1
560 IF Y>24 THEN Y=1
570 RETURN
580 X=X-1 :: IF X<3 THEN X=30 :: Y=Y-1
590 IF Y<1 THEN Y=24
600 RETURN
610 Y=Y+1 :: IF Y>24 THEN Y=1
620 RETURN
630 Y=Y-1 :: IF Y<1 THEN Y=24
640 RETURN
650 CALL DELSPRITE(ALL):: CALL SCREEN(9):: CALL COLOR(13,9,1,14,9,1)
660 FOR I=1 TO 24
670 FOR K=3 TO 30
680 CALL GCHAR(I,K,MASKE):: MASKE$(I)=MASKE$(I)&CHR$(MASKE)
690 NEXT K :: NEXT I
700 FOR I=1 TO 24 :: DISPLAY AT(I,1):" " :: NEXT I
710 CALL SCREEN(16):: CALL COLOR(13,16,1,14,16,1)
720 FOR I=1 TO 24 :: DISPLAY AT(I,1):MASKE$(I):: NEXT I
730 ON SKIP GOTO 740,1670
740 ! Attributbildschirm
750 X=3 :: Y=1
760 CALL SPRITE(£2,65,5,8,8)
770 T=13 :: GOSUB 500
780 CALL KEY(5,T,S):: IF T>7 AND T<12 THEN GOSUB 500
790 IF T=60 XOR T=62 THEN 830
800 IF T=13 THEN CALL SOUND(-100,262,5):: GOSUB 500
810 IF T=7 THEN 860
820 IF T=1 THEN 870 ELSE 780
```

```
830 CALL HCHAR(Y,X,T)
840 T=9 :: CALL SOUND(-100,523,5):: GOSUB 500
850 GOTO 780
860 CALL HCHAR(Y,X,32):: SIGNALT=32 :: T=9 :: CALL SOUND(-100,131,5):: GOSUB 500
 :: GOTO 780
870 CALL DELSPRITE(ALL):: CALL COLOR(13,9,1,14,9,1):: CALL SCREEN(9)
880 SUCH=1
890 SKIP=0
900 FOR Y=1 TO 24
910 FOR X=3 TO 30
920 CALL GCHAR(Y,X,WERT)
930 IF WERT=60 OR WERT=62 THEN GOSUB 1810
940 IF SKIP=1 THEN GOTO 1770
950 NEXT X
960 NEXT Y
970 IF FA(SUCH)<>0 THEN 1770
980 CALL CLEAR :: CALL VCHAR(1,1,128,48):: CALL VCHAR(1,31,128,48):: CALL COLOR(
13,2,9,14,2,9):: PRINT "<S-D> generiert die Felder": :
990 CALL SOUND(-100,523,5)
1000 FOR SUCH=1 TO 25
1010 IF FA(SUCH)=0 THEN 1090
1020 WA=FA(SUCH)-INT(FA(SUCH)):: WA=WA*100 :: WE=FE(SUCH)-INT(FE(SUCH)):: WE=WE*
100
1030 USER$(SUCH)=SEG$(MASKE$(INT(FA(SUCH))),WA,FELD(SUCH))
1040 B$=USER$(SUCH)
1050 FOR K=1 TO LEN(USER$(SUCH))
1060 IF ASC(B$)=35 XOR ASC(B$)=46 THEN 1070 ELSE GOSUB 1760
1070 B$=SEG$(B$,2,LEN(B$))
1080 NEXT K :: NEXT SUCH
1090 PRINT "<S-D> loescht die Felder in       der Bildmaske": : :: CALL SOUND(-1
00,523,5)
1100 FOR SUCH=1 TO 25
1110 IF FA(SUCH)=0 THEN 1160
1120 W1=INT(FA(SUCH)):: W2=FA(SUCH)-INT(FA(SUCH)):: W2=W2*100
1130 FOR DEL=1 TO FELD(SUCH)
1140 MASKE$(W1)=SEG$(MASKE$(W1),1,W2-2+DEL)&" "&SEG$(MASKE$(W1),W2+DEL,28)
1150 NEXT DEL :: NEXT SUCH
1160 CALL CLEAR :: CALL COLOR(13,16,2,14,16,2):: CALL SCREEN(2):: CALL VCHAR(1,1
,128,48):: CALL VCHAR(1,31,128,48)
1170 CALL HCHAR(4,3,128,28)
1180 CALL HCHAR(21,3,128,28)
1190 FOR SUCH=1 TO 25
1200 IF FA(SUCH)=0 THEN 1410
1210 DISPLAY AT(2,1):MASKE$(INT(FA(SUCH))):: DISPLAY AT(2,(FA(SUCH)-INT(FA(SUCH)
))*100)SIZE(FELD(SUCH)):USER$(SUCH)
1220 DISPLAY AT(6,1):"Felder vor Ausgabe loeschen (J/N)   --->   <---"
1230 DISPLAY AT(9,1):"Akkustisches Signal bei der Ausgabe des Feldes          (J
/N)   --->   <---"
1240 DISPLAY AT(23,7):USING "Feld Nr.: ##":SUCH
1250 IF SGN(FELD(SUCH))=-1 THEN DISPLAY AT(7,14)SIZE(1):"N" ELSE DISPLAY AT(7,14
)SIZE(1):"J"
1260 ACCEPT AT(7,14)VALIDATE("JN")SIZE(-1)BEEP:X$ :: IF X$="N" THEN FELD(SUCH)=-
FELD(SUCH)
1270 IF TON(SUCH)=1 THEN DISPLAY AT(11,14)SIZE(1):"J" ELSE DISPLAY AT(11,14)SIZE
(1):"N"
1280 ACCEPT AT(11,14)VALIDATE("JN")SIZE(-1)BEEP:X$ :: IF X$="J" THEN TON(SUCH)=1
 ELSE TON(SUCH)=0
1290 DISPLAY AT(13,1):"Typ der Variablen            0 - keine Vereinbarung"
1300 DISPLAY AT(15,1):"1 - UALPHA (Grossbuchstaben)2 - DIGIT  (Ziffern 1 bis 9)3
 - NUMERIC(Ziffen,' .+-E ')4 - eigener Text"
1310 DISPLAY AT(19,9):"--->   <---"
1320 IF LEN(VALD$(SUCH))>1 THEN DISPLAY AT(19,14)SIZE(1):"4" :: VA$=VALD$(SUCH)E
```

```
LSE DISPLAY AT(19,14)SIZE(1):VALD$(SUCH)
1330 ACCEPT AT(19,14)SIZE(-1)VALIDATE("01234")BEEP:VALD$(SUCH)
1340 IF VALD$(SUCH)="" OR VALD$(SUCH)=" " THEN 1330
1350 IF VALD$(SUCH)="4" THEN 1360 ELSE 1390
1360 DISPLAY AT(19,1):"Text : "
1370 DISPLAY AT(19,9):VA$
1380 ACCEPT AT(19,9)SIZE(-20)BEEP:VALD$(SUCH)
1390 FOR I=13 TO 19 :: DISPLAY AT(I,1):"" :: NEXT I
1400 NEXT SUCH
1410 GOSUB 1430
1420 GOTO 1540
1430 CALL CLEAR
1440 CALL VCHAR(1,31,128,96)
1450 FOR I=1 TO 24
1460 DISPLAY AT(I,1):MASKE$(I)
1470 NEXT I
1480 FOR SUCH=1 TO 25
1490 IF FA(SUCH)=0 THEN 1530
1500 W1=FA(SUCH)-INT(FA(SUCH)):: W1=W1*100 :: W2=INT(FA(SUCH))
1510 DISPLAY AT(W2,W1)SIZE(FELD(SUCH)):USER$(SUCH)
1520 NEXT SUCH
1530 RETURN
1540 CALL KEY(5,T,S):: IF T<>15 THEN IF T<>12 THEN 1540
1550 IF T=15 THEN SKIP=2 :: GOTO 310
1560 CALL CLEAR :: DISPLAY AT(12,1):"Name der Datei, worauf die  Maske gespeiche
rt werden    soll ?  (Geraet.Name)"
1570 ACCEPT AT(15,1)BEEP:DATEI$
1580 OPEN £2:DATEI$,OUTPUT,SEQUENTIAL,INTERNAL,FIXED 80
1590 FOR I=1 TO 24 :: PRINT £2:MASKE$(I):: NEXT I
1600 FOR I=1 TO 25 :: PRINT £2:FELD(I),TON(I),FA(I),VALD$(I),USER$(I)
1610 IF FA(I)=0 THEN 1630
1620 NEXT I
1630 CALL CLEAR
1640 ON ERROR STOP
1650 CLOSE £2
1660 STOP
1670 FOR I=1 TO 25
1680 IF FA(I)=0 THEN 1730
1690 ZA=INT(FA(I)):: SA=(FA(I)-INT(FA(I)))*100
1700 DISPLAY AT(ZA,SA-1)SIZE(1):"<"
1710 DISPLAY AT(ZA,SA+ABS(FELD(I)))SIZE(1):">"
1720 NEXT I
1730 FOR I=1 TO 25 :: FA(I),FE(I)=0 :: USER$(I)="" :: NEXT I
1740 FA(0),FE(0)=1
1750 GOTO 740
1760 USER$(SUCH)=SEG$(USER$(SUCH),1,K-1)&"£"&SEG$(USER$(SUCH),K+1,LEN(USER$(SUCH
))-K):: RETURN
1770 CALL SPRITE(£2,70,16,8,8)
1780 CALL SPRITE(£1,96,16,Y*8-7,X*8-7):: CALL SOUND(1000,262,0,330,0,196,0)
1790 FOR I=1 TO 25 :: FA(I),FE(I),FELD(I)=0 :: NEXT I
1800 GOTO 780
1810 !
1820 IF WERT=62 THEN 1860
1830 IF FA(SUCH)<>0 THEN SKIP=1 :: RETURN
1840 FA(SUCH)=Y+(X-1)/100
1850 RETURN
1860 IF FA(SUCH)=0 THEN SKIP=1 :: RETURN
1870 IF INT(FA(SUCH))<>Y THEN SKIP=1 :: RETURN
1880 FE(SUCH)=Y+(X-2)/100
1890 W1=FE(SUCH)-INT(FE(SUCH)):: W1=W1*100 :: W2=FA(SUCH)-INT(FA(SUCH)):: W2=W2*
100
1900 FELD(SUCH)=W1-W2
```

```
1910 IF FELD(SUCH)<1 OR FELD(SUCH)>14 THEN SKIP=1 :: RETURN
1920 SUCH=SUCH+1
1930 IF SUCH>25 THEN SKIP=1 :: RETURN
1940 !
1950 RETURN
1960 DISPLAY AT(15,1):"Welche Maske soll veraendertwerden ?   (Geraet.name)"
1970 ACCEPT AT(17,1)BEEP:DATEI$
1980 ON ERROR 1970
1990 OPEN £1:DATEI$,INPUT ,SEQUENTIAL,INTERNAL,FIXED 80
2000 FOR I=1 TO 24 :: INPUT £1:MASKE$(I):: NEXT I
2010 FOR I=1 TO 25 :: INPUT £1:FELD(I),TON(I),FA(I),VALD$(I),USER$(I)
2020 IF FA(I)=0 THEN 2040
2030 NEXT I
2040 CLOSE £1
2050 ON ERROR STOP
2060 GOTO 2170
2070 FOR I=2 TO 12 :: CALL COLOR(I,2,11):: NEXT I
2080 CALL CLEAR :: CALL VCHAR(1,31,128,96):: CALL COLOR(1,2,11,13,11,5):: CALL S
CREEN(5)
2090 CALL MAGNIFY(2):: CALL SPRITE(£1,83,7,17,60,£2,68,7,17,132)
2100 DISPLAY AT(4,8):"creen    esigner"
2110 DISPLAY AT(23,4):"(c) 1984 by Tronicsoft"
2120 DISPLAY AT(9,1):"Bitte waehlen Sie aus --->"
2130 DISPLAY AT(15,1):"1 - Neue Maske erstellen"
2140 DISPLAY AT(17,1):"2 - Vorhandene Maske aendern"
2150 ACCEPT AT(9,28)VALIDATE("12")BEEP:SKIP
2160 ON SKIP GOTO 2170,1960
2170 CALL DELSPRITE(ALL):: CALL CLEAR :: CALL MAGNIFY(1):: GOTO 310
```

Unterprogramm SCREEN READER

Um eine mit SCREEN DESIGNER erstellte Maske im eigenen Extended-BASIC-Programm zu verwenden, wird das Unterprogramm MASKE, der sogenannte SCREEN READER benutzt.

Benutzer eines Kassettenrecorders schreiben das Programm separat auf Kassette weg, und laden dieses, bevor sie ihr eigenes Programm schreiben, ein. Benutzer eines Diskettenlaufwerkes speichern das Programm im MERGE Modus ab (z. B. SABE DKS1.MASKE,MERGE) und können dieses dann mit MERGE auf jedes beliebige Programm aufladen. Dabei ist lediglich zu beachten, daß die Programmzeilen ab 32000 nicht vom eigenen Programm belegt sind.

Doch nun zum Programm:

Zeile 32060 hat die allgemeine Form:
DIM MASKE$(n,24), VALD$(n,25), SZE(n,25), DAT(n,25), USER$(n,25), Ton(n,25)
wobei n die Anzahl der gleichzeitig verwaltbaren Masken angibt. Die Anzahl ist lediglich durch den zur Verfügung stehenden Speicherplatz begrenzt.

Im Programm ist n mit 1 angegeben. Das bedeutet, daß maximal 2 Masken gleichzeitig verwaltet werden können (Maske Nr. 0 und Maske Nr. 1). Die Anzahl kann jedoch vom Benutzer beliebig erhöht werden.

Der Aufruf des Unterprogramms hat folgende Form:
CALL MASKE (ACT,MNR,FNR, numerische Variable oder Zahl, Stringvariable oder Text).

Hierin bedeuten:

ACT – Eine Anzahl zwischen 1 und 5, welche angibt, was mit, bzw. in der Maske geschehen soll:

1 – Die Maske wird auf dem Bildschirm ausgegeben.

2 – Eine Variable wird in der Maske eingelesen.

3 – Eine Variable, bzw. Zahl oder Text wird in einem Feld der Maske ausgegeben.

4 – Eine Maske wird von Kassette oder Diskette geladen.

5 – Das Unterprogramm wird für eine oder mehrere Masken initialisiert, d. h., es wird der Speicherplatz für die Masken zur Verfügung gestellt.

MNR – Gibt die Nummer der Maske an, mit welcher die entsprechende Aktion ausgeführt werden soll.

FNR – Gibt die Feldnummer in der entspr. Maske an, in welcher die Aktion durchgeführt werden soll.

Numerische Variable – ist eine Variable nach Ihrer Wahl.

Stringvariable – ist eine Textvariable nach Ihrer Wahl.

Die ganze Maskenverwaltung ist relativ einfach mit einem einzigen Befehl zu bearbeiten.

Wir wollen dies an einem Beispiel verdeutlichen:

Nehmen wir an, Sie haben 2 Masken, welche Sie verwalten wollen, und nehmen wir an, diese Masken sind auf Kassette gespeichert.

Zuerst müssen wir die Maskenverwaltung initialisieren und geben den Befehl: CALL MASKE(5,0,0,V,V$) ein. Die DIM-Anweisung in Zeile 32060 wurde nicht verändert, und der Platz für 2 Masken ist jetzt reserviert.

Dieser Befehl muß als erster Maskenbefehl stehen, und darf im weiteren Programmverlauf nicht noch einmal vorkommen.

Jetzt laden wir die erste Maske ein. Dazu geben wir den Befehl: CALL MASKE(4,0,0,V,V$) und der Rechner fragt nach dem Eingabegerät. Tippen Sie CSi ein, und drücken Sie danach ⟨ENTER⟩. Der Rechner lädt die Maske unter der Nummer 0 ein.

Zum Einlesen der 2. Maske geben wir den Befehl CALL MASKE(4,1,0,V,V$) ein, und wieder werden wir nach dem Eingabegerät gefragt. Geben Sie wieder CSi ein und drücken Sie ⟨ENTER⟩, um die 2. Maske zu laden. Die beiden Masken sind nun im Rechner gespeichert.

Wir rufen jetzt Maske 2 auf: CALL MASKE(1,1,0,V,V$), und die Maske wird auf dem Bildschirm ausgegeben. Nehmen wir an, die Maske hat 15 Felder, und aus allen Feldern soll eine Variable eingelesen werden. Dazu haben wir zuvor im Programm ein Feld mit dem Namen EINGABE dimensioniert.
Die Eingabe der Felder kann nun wie folgt erfolgen:
CALL MASKE(2,1,I,EINGABE(I),V$),
NEXT I.
Sie haben somit alle 15 Felder der Maske eingegeben, wobei es völlig egal ist, wo die Felder auf dem Bildschirm liegen.
Der SCREEN READER vereinbart die Felder wie folgt:
Felder, die den Variablentyp DIGIT oder NUMERIC haben, werden in die numerische Variable eingelesen.
Felder mit den Variablentypen UALPHA, eigener Text oder wenn keine Vereinbarung getroffen wurde, werden in die Stringvariable des Parameters eingelesen.

Doch nun weiter im Beispiel:

Wir geben jetzt unsere Maske 1 auf dem Bildschirm mit dem Befehl CALL MASKE(1,0,0,V,V$) aus.
Das Feld 3 haben wir im SCREEN DESIGNER als Stringfeld ausgewiesen, und wollen hier den Kommentar Computronic ausgeben. Dazu schreiben wir den Befehl CALL MASKE(3,0,3,V,„Computronic") und der Text erscheint im Feld 3.
Die Maskenverwaltung geht mit diesem Unterprogramm wesentlich komfortabler und einfacher als mit dem ewigen DISPLAY AT und ACCEPT AT. Die Verarbeitung ist weiterhin auch platzsparender, da man im Verlauf eines Programmes jederzeit eine nicht mehr benötigte Maske durch eine andere Maske überschreiten kann.
Nehmen wir an, wir brauchen unsere Maske 2 nicht mehr, so können wir mit CALL MASKE(4,1,0,V,V$) eine neue Maske in den Speicherbereich der Maske 2 laden. Wird dann die Maske 2 angesprochen, so erhalten wir die neu eingeladene Maske auf den Bildschirm.
Es gibt viele Möglichkeiten der Maskenverwaltung, die in dieser kurzen Beschreibung nicht alle abgehandelt werden können. Am besten Sie probieren die Sache einfach selbst aus.
Es ist jedoch darauf zu achten, daß sie keine Maskennummer ansprechen, die außerhalb der durch DIM in Zeile 32060 liegenden Dimension liegt. Ferner dürfen Sie keine Feldnummer ansprechen, die nicht im SCREEN DESIGNER definiert wurde.
Ein Versuch dieses zu tun, wird mit einer Fehlermeldung und mit Programmbruch quittiert.
Es gehört sicher etwas Übung zu dieser neuartigen Programmtechnik, jedoch werden Sie dieses ganz schnell in den Griff bekommen, und dann den Maskengenerator nicht mehr missen wollen.
Mit dem Programm SCREEN PRINTER können Sie sich eine Aufschlüsselung einer durch SCREEN DESIGNER erstellten Maske auf dem Drukker ausdrucken lassen. Sie haben somit mit einem Blick die Kontrolle, wie die Maske aufgebaut ist, und wo die einzelnen Felder liegen. Ferner bekommen Sie sämtliche Optionen der einzelnen Felder ausgedruckt.
Als Anlage ist ein Probeausdruck einer Maske beigefügt.
Die im Feld „Länge" ausgebenen Werte sind wie folgt zu lesen:
Postitive Länge – Das Feld wird vor Eingabe gelöscht.
Negative Länge – Das Feld wird nicht gelöscht.
Laden Sie das Programm mit RUN „CS1" oder mit RUN „DSK1.S-PRINTER" ein. Es startet sich jetzt von selbst.
Geben Sie den Namen der aufzuschlüsselnden Maske ein (z. B. CS1 oder DSK1.MASKE).
Sie werden jetzt nach dem Druckertyp gefragt. Hierbei antworten Sie dem Rechner mit der entsprechenden Schnittstellenoption, an welche Sie Ihren Drucker angeschlossen haben (z. B. RS232 für die serielle Schnittstelle, oder PIO für die parallele Schnittstelle).
Danach geben Sie dem Rechner noch das aktuelle Datum ein, und der Ausdruck beginnt.

ANMERKUNG:

Das Programm ist auf einen SEIKOSHA-Drucker der GP-Serie abgestimmt. Sollten Sie einen anderen Drucker besitzen, so müssen Sie möglicherweise die Programmzeile 620 Print #1:CHR$(15) abändern, denn dieses Steuerzeichen schaltet den Drucker von Breitschrift auf Normalschrift zurück.
So muß die Zeile z. B. bei Verwendung eines STAR Druckers PRINT #1: CHR$(20) heißen.

```
32000 SUB MASKE(ACT,MNR,FNR,VAR,VAR$)
32010 REM Unterprogramm SCREEN READER
32020 REM Maskenverwaltungsprogramm zu SCREEN DESIGNER
32030 REM (c) 1984 by Tronicsoft
32040 STV=0
32050 ON ACT GOTO 32370,32090,32110,32270,32060
32060 DIM MASKE$(1,24),VALD$(1,25),SZE(1,25),DAT(1,25),USER$(1,25),TON(1,25)
32070 ON WARNING NEXT
32080 SUBEXIT
32090 STV=1
32100 IF TON(MNR,FNR)=1 THEN CALL SOUND(-50,698,0)
32110 IF LEN(VALD$(MNR,FNR))>1 THEN 32210
32120 IF VAL(VALD$(MNR,FNR))=0 THEN 32210
32130 ON VAL(VALD$(MNR,FNR))GOTO 32210,32140,32140
32140 DISPLAY AT(INT(DAT(MNR,FNR)),(DAT(MNR,FNR)-INT(DAT(MNR,FNR)))*100)SIZE(SZE
(MNR,FNR)):USING USER$(MNR,FNR):VAR
32150 IF STV=0 THEN SUBEXIT
32160 ON VAL(VALD$(MNR,FNR))GOTO 32390,32170,32190
32170 ACCEPT AT(INT(DAT(MNR,FNR)),(DAT(MNR,FNR)-INT(DAT(MNR,FNR)))*100)SIZE(SZE(
MNR,FNR))VALIDATE(DIGIT):VAR
32180 SUBEXIT
```

```
32190 ACCEPT AT(INT(DAT(MNR,FNR)),(DAT(MNR,FNR)-INT(DAT(MNR,FNR)))*100)SIZE(SZE(
MNR,FNR))VALIDATE(NUMERIC):VAR
32200 SUBEXIT
32210 DISPLAY AT(INT(DAT(MNR,FNR)),(DAT(MNR,FNR)-INT(DAT(MNR,FNR)))*100)SIZE(SZE
(MNR,FNR)):USING USER$(MNR,FNR):VAR$
32220 IF STV=0 THEN SUBEXIT
32230 IF VALD$(MNR,FNR)="1" THEN ACCEPT AT(INT(DAT(MNR,FNR)),(DAT(MNR,FNR)-INT(D
AT(MNR,FNR)))*100)SIZE(SZE(MNR,FNR))VALIDATE(UALPHA):VAR$
32240 IF VALD$(MNR,FNR)="0" THEN ACCEPT AT(INT(DAT(MNR,FNR)),(DAT(MNR,FNR)-INT(D
AT(MNR,FNR)))*100)SIZE(SZE(MNR,FNR)):VAR$
32250 IF LEN(VALD$(MNR,FNR))>1 THEN ACCEPT AT(INT(DAT(MNR,FNR)),(DAT(MNR,FNR)-IN
T(DAT(MNR,FNR)))*100)SIZE(SZE(MNR,FNR))VALIDATE(VALD$(MNR,FNR)):VAR$
32260 SUBEXIT
32270 CALL CLEAR :: DISPLAY AT(1,1)BEEP:"Name der zu ladenden Maske :(Geraet.Dat
ei)" :: ACCEPT AT(3,1):DATEI$
32280 ON ERROR 32270
32290 OPEN £99:DATEI$,INPUT ,SEQUENTIAL,INTERNAL,FIXED 80
32300 FOR I=1 TO 24 :: INPUT £99:MASKE$(MNR,I):: NEXT I
32310 FOR I=1 TO 25 :: INPUT £99:SZE(MNR,I),TON(MNR,I),DAT(MNR,I),VALD$(MNR,I),U
SER$(MNR,I)
32320 IF DAT(MNR,I)=0 THEN 32340
32330 NEXT I
32340 CLOSE £99 :: CALL CLEAR
32350 ON ERROR STOP
32360 SUBEXIT
32370 CALL CLEAR
32380 FOR I=1 TO 24 :: DISPLAY AT(I,1):MASKE$(MNR,I):: NEXT I
32390 SUBEND
```

Ein Fehler kann auftreten, wenn:

1. Ein Feldende Attribut gesetzt wurde, ohne daß ein Feldanfang Attribut gesetzt wurde.
2. Ein zweiter Feldanfang definiert wurde, ohne daß das vorherige Feld geschlossen wurde.
3. Ein Feld länger als 14 Zeichen ist.
4. Ein Feld in einer Zeile beginnt und erst in der nächsten Zeile endet.
5. Das letzte Feldende Attribut vergessen wurde. In diesem Fall erscheint der Cursor nicht, wird jedoch nach der Betätigung einer Cursorsteuertaste in der unteren rechten Ecke sichtbar.

Steuern Sie im Falle eines Fehlers den Cursor an die entsprechende Stelle, und ändern Sie diese, wie im Attributbildschirm beschrieben, ab. Danach drücken Sie wieder „AID", um einen neuen Fehlertestlauf zu starten.

Wenn keine Fehler im Attributbildschirm aufgetreten sind, so beginnt der Rechner mit der eigentlichen Maskengenerierung. Die Aktionen werden Ihnen auf dem Bildschirm mitgeteilt.

Der Rechner generiert in dieser Zeit das DISPLAY AT Format, die Feldlänge SIZE(), sowie das USING Format.

Nachdem die Generierung abgeschlossen ist, kommen Sie zur Variablenvereinbarung:

Im ersten Feld wird gefragt, ob das entsprechende Feld vor einer Eingabe am Bildschirm gelöscht werden soll. Die Standardvorgabe hierbei ist immer „J". Wenn Sie später eine Variable jedoch nur in der Maske ändern wollen, so antworten Sie dem Rechner mit „N", denn dann bleibt die Variable auf dem Bildschirm sichtbar und kann beliebig geändert werden.

Bei der Frage nach dem akustischen Signal ist die Standardoption einer neuen Maske „N", bei einer zu editierenden Maske wird die Option der alten Maske ausgegeben. Wenn Sie also einen Piepston bei der Feldeingabe wünschen, so antworten Sie mit „J".

Als nächstes kommt die Variablenvereinbarung an die Reihe. Sie haben die Auswahl zwischen 5 Möglichkeiten:

Auswahl 0 bedeutet keine Vereinbarung. Es entfällt bei diesem Feld also das SIZE Format.

Auswahl 1 läßt bei Eingabe nur Großbuchstaben zu.

Auswahl 2 läßt bei der Eingabe nur Zahlen zu.

Auswahl 3 läßt bei der Eingabe nur Zahlen sowie die Zeichen + − . E zu.

Bei Auswahl 4 können Sie selbst wählen, welche Zeichen und Zahlen bei der Eingabe zugelassen werden. Wenn Sie eine Maske ändern, so werden die Optionen der alten Maske ausgegeben, können jedoch beliebig geändert werden.

Der Rechner läuft alle Felder durch und fragt nach den Variablenvereinbarungen. Danach wird die generierte Maske ausgegeben. Sie sehen den Text und die USING Formate an den entsprechenden Bildschirmstellen.

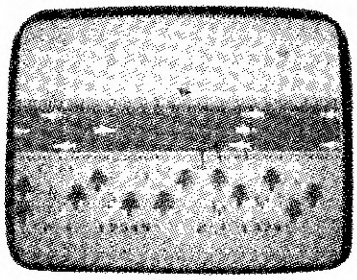
Wollen Sie die Maske jetzt noch ändern, so drücken Sie die Funktionstaste BACK (FCTN 9), und Sie kommen wieder in den Leerbildschirm, allerdings wird die Maske bereits auf dem Bildschirm ausgegeben, und der Leerbildschirm hat jetzt schwarze Randstreifen. Sind Sie mit der Maske zufrieden, so drücken Sie die Funktionstaste PROC'D (FCTN 6). Sie werden nun gefragt, wo die Maske gespeichert werden soll. Geben Sie das entsprechende Gerät ein (z. B. CS1 oder DSK1.MASKE), und der Rechner speichert die Maske ab. Der Speichervorgang dauert je nach Anzahl der Felder mit dem Cassettenrecorder ca. 4 bis 8 Minuten, mit einem Diskettenlaufwerk ca. 8 bis 12 Sekunden.

Lassen Sie sich von der Anleitung nicht beirren, denn die Maskenerstellung ist relativ einfach. Es hört sich alles viel schwieriger an, als es in Wirklichkeit ist. Probieren Sie es einfach einmal aus, und denken Sie daran, daß ein Programm niemals den Computer kaputtmachen kann. Es kann höchstens zum Systemabsturz kommen, was sich jedoch mit Ausschalten ohne weiteres wieder beheben läßt.

```
100 !@P-
110 REM ********************
120 REM *  SCREEN  PRINTER  *
130 REM *--------------------*
140 REM *  fuer TI 99 / 4A  *
150 REM *  + Extended BASIC *
160 REM *  + ext. Speicher  *
170 REM *--------------------*
180 REM * (c)1984 Tronicsoft*
190 REM *          V.Becker  *
200 REM ********************
210 REM
220 REM
230 !@P+
240 DIM MASKE$(24),VALD$(25),SZE(25),DAT(25),USER$(25),TON(25)
250 GOTO 270 :: DATEI$ :: DATUM$ :: DRUCK$ :: I :: K :: P :: W1 :: W2 :: CALL CL
EAR :: CALL GCHAR :: CALL SCREEN
260 !@P-
270 CALL SCREEN(4)
280 DISPLAY AT(1,1)ERASE ALL:"S C R E E N    P R I N T E RAufschluesselung einer
 durch SCREEN DESIGNER erstellten        Bilschirmmaske        ------------------
-----------"
290 DISPLAY AT(7,1):"Name der aufzuschluesselndenMaske : (Geraet.name)"
300 ACCEPT AT(9,1)BEEP:DATEI$
310 ON ERROR 300
320 OPEN #1:DATEI$,INPUT ,SEQUENTIAL,INTERNAL,FIXED 80
330 FOR I=1 TO 24 :: INPUT #1:MASKE$(I):: NEXT I
340 FOR I=1 TO 25 :: INPUT #1:SZE(I),TON(I),DAT(I),VALD$(I),USER$(I)
350 IF DAT(I)=0 THEN 370
360 NEXT I
370 CLOSE #1
380 ON ERROR STOP
390 DISPLAY AT(12,1):"Drucker : " :: ACCEPT AT(12,11)BEEP:DRUCK$
400 ON ERROR 390
410 OPEN #1:DRUCK$
420 ON ERROR STOP
430 DISPLAY AT(15,1)BEEP:"Heutiges Datum : " :: ACCEPT AT(15,18)VALIDATE(NUMERIC
):DATUM$
440 PRINT #1:"     Aufschluesselung der Maske : ";DATEI$;"     ";DATUM$
450 FOR I=1 TO 80 :: PRINT #1:"-";:: NEXT I :: PRINT #1
460 CALL CLEAR
470 FOR I=1 TO 24 :: DISPLAY AT(I,1):MASKE$(I):: NEXT I
480 FOR I=1 TO 25
490 IF DAT(I)=0 THEN 530
500 W1=(DAT(I)-INT(DAT(I)))*100 :: W2=INT(DAT(I))
510 DISPLAY AT(W2,W1)SIZE(SZE(I)):USER$(I)
520 NEXT I
530 PRINT #1 :: PRINT #1 :: PRINT #1:CHR$(14)
540 PRINT #1:"       12345678901234567890123456 78"
550 PRINT #1:"      ------------------------------"
560 FOR I=1 TO 24
570 PRINT #1:"   ";:: PRINT #1,USING "##":I;:: PRINT #1:":";
580 FOR K=3 TO 30 :: CALL GCHAR(I,K,P):: PRINT #1:CHR$(P);:: NEXT K
590 PRINT #1:":"
600 NEXT I
610 PRINT #1:"      ------------------------------"
620 PRINT #1:CHR$(15)! Umschaltung auf Normalschrift : je nach Drucker bitte aba
endern
630 PRINT #1
640 PRINT #1:"    Feld Nr. : Zeile : Spalte : Laenge : Akkustisches Signal : Var
iablen Typ"
650 FOR I=1 TO 80 :: PRINT #1:"-";:: NEXT I :: PRINT #1
```

```
660 FOR I=1 TO 25
670 IF DAT(I)=0 THEN 800
680 PRINT £1:"        ";:: PRINT £1,USING "££":I;:: PRINT £1:"    :    ";
690 PRINT £1,USING "££":INT(DAT(I));:: PRINT £1:"   :    ";:: PRINT £1,USING "££":
(DAT(I)-INT(DAT(I)))*100;
700 PRINT £1:"    :   ";:: PRINT £1,USING "£££":SZE(I);:: PRINT £1:"    :          "
;
710 IF TON(I)=1 THEN PRINT £1,USING " ££ ":"ja";ELSE PRINT £1,USING "££££":"nein
";
720 PRINT £1:"          : ";
730 IF LEN(VALD$(I))>1 THEN PRINT £1:VALD$(I):: GOTO 790
740 IF VALD$(I)="0" THEN PRINT £1:"keine Vereinb." :: GOTO 790
750 ON VAL(VALD$(I))GOTO 760,770,780
760 PRINT £1:"UALPHA" :: GOTO 790
770 PRINT £1:"DIGIT" :: GOTO 790
780 PRINT £1:"NUMERIC"
790 NEXT I
800 FOR I=1 TO 80 :: PRINT £1:"-";:: NEXT I
810 PRINT £1 :: PRINT £1
820 PRINT £1:"    Ende der Aufschluesselun9"
830 CLOSE £1
840 CALL CLEAR
850 STOP
860 END
```

Mother Duck



Sind Sie schon einmal in die Rolle einer Entenmutter geschlüpft, die verzweifelt versucht, ihre Kinder zu füttern?

Wenn nicht, so können Sie es mit diesem Programm tun. Sie brauchen hierzu einen TI 99/4 A Homecomputer, das TI Extended BASIC Modul, einen Joystick und einen Cassettenrecorder, um das Programm abzuspeichern, denn Sie wollen ja sicherlich nicht vor jedem Spiel das komplette Programm eintippen.

Helfen Sie der Entenmutter, Ihre Kinder zu füttern

Tippen Sie das Programm ein und sichern Sie dieses dann auf Cassette.
Zum Spiel legen Sie nun die Cassette ein und starten mit RUN „CS1".
Kurze Zeit später erscheint auf dem Bildschirm das Titelbild mit der Frage: BEGINN BEI RUNDE ‹1–10›. Der Computer spielt Ihnen hierbei eine Melodie vor, bis Sie auf dem Keyboard irgend eine Taste drücken. Haben Sie jetzt eine Taste gedrückt, so erscheint der blinkende Cursor und Sie können die Runde eingeben, bei welcher Sie beginnen wollen.
Geben Sie eine Runde größer 10 ein, so beginnt das Spiel bei Runde 10.
Haben Sie die Runde eingegeben, so baut der Rechner das Spielfeld auf. Dies kann jedoch bei steigender Rundenzahl ein klein wenig dauern, da das Programm rein in BASIC geschrieben ist und beim Aufbau des unteren Bildschirmteiles (Wald) eine Menge Bildschirmvergleiche durchgeführt werden müssen.
Wenn das Bild nun generiert ist, so erscheint in der Mitte des oberen Flußufers unsere Entenmutter und der Computer spielt uns SALOME vor.
Um die Runde zu beginnen, drücken wir den roten Aktionsknopf am Joystick.

Ziel des Spieles

Die Ente muß über den Fluß in den Wald geführt werden, um dort die Würfel aufzusammeln.
Diese Würfel muß sie wieder über den Fluß auf die obere Seite des Bildschirmes bringen, um damit eine Pyramide aufbauen zu können.
Nur so kann unsere Entenmutter die Kirsche oben rechts im Bild erreichen.
Die Ente kann jedoch immer nur einen Würfel tragen. Hat unsere Ente nun die Kirsche erreicht, so muß sie diese zum Nest mit ihren Kindern bringen.
Das alles hört sich jedoch einfacher an, als es ist, denn es gibt viele Möglichkeiten, wie unsere Ente das Leben verlieren kann.
Wenn unsere Ente den Fluß überquert, so sollte sie sich am besten mit keinem Schwertfisch einlassen, denn dieser Schwertfisch könnte schlecht gelaunt sein, und unserer Ente den Garaus machen.
Die Ente bewegt sich im Wasser schneller als auf dem Land. Deshalb ist es ratsam, wenn die Ente den Fluß in Richtung Wald verläßt, dies nicht im Bereich eines Würfels zu tun, da es unserer Ente durch den Aufprall ein Leben kosten könnte.
Unsere Ente darf auf dem Landweg den Bildschirm nicht verlassen, da sie sonst stirbt.
Auch sollte die Ente nicht versuchen, durch einen Baum zu gehen, sie könnte sich verletzen und im Wald sterben.
Wenn unsere Ente endlich die Kirsche erreicht hat, so kommt aus dem Wald der böse Schrat und versucht, ihr die Kirsche abzunehmen.

Gelingt dem bösen Schrat dieses, so verliert unsere Ente ein Leben.
In dieser Spielphase darf die Ente auch nicht in den Fluß, da sie sonst ertrinkt. Steuern Sie die Ente zum Nest.

Steuerung

Die Ente wird mit dem Joystick in 8 Richtungen bewegt. Zum Aufnehmen eines Würfels gehen Sie mit der Ente über den Würfel und drücken Sie den Aktionsknopf. Wenn Sie mit dem Würfel die andere Seite des Flußufers erreicht haben, so wird der Würfel automatisch gutgeschrieben. Es ertönt ein Glockenschlag und der Würfel erscheint oben links im Bild.
Nachdem alle 10 Würfel gesammelt wurden, erscheint die Pyramide unter der Kirsche und die Ente sitzt ganz oben auf der Pyramide.
Drücken Sie jetzt den Aktionsknopf, um fortzufahren. Der Schrat erscheint und verfolgt die Ente. Führen Sie die Ente zum Nest mit den Kindern.
Wenn Sie das Nest erreicht haben, so ertönt wieder der Glockenschlag, und es geht mit der nächsten Runde weiter. Jedesmal, wenn sie SALOME hören, müssen Sie den Aktionsknopf drücken, um die nächste Runde zu beginnen.

Punktwertung

Die Punktanzeige ist unten links im Bild eingeblendet und mit „P" gekennzeichnet.
Rechts daneben ist mit „Z" gekennzeichnet der Zeitbonus eingeblendet.
Dieser Zeitbonus hat zu Beginn jeder Runde den Wert 2000 und verringert sich im Laufe des Spieles.
Hat der Zeitbonus den Wert 0 erreicht, so stirbt die Entenmutter eines ihrer 5 Leben.
Wenn Sie die Kirsche erreicht haben, so erhalten Sie 1000 Punkte.
Haben Sie die Kirsche den Entenkindern gebracht, so wird der verbleibende Zeitbonus Ihrem Punktekonto gutgeschrieben.
Für jeden Würfel, den Sie gesammelt haben und auf die andere Seite des Flusses gebracht haben, erhalten Sie je nach Farbe:

weiß: 20 Punkte
blau: 40 Punkte
rot: 60 Punkte
grün: 80 Punkte

auf Ihrem Punktekonto gutgeschrieben.
In der oberen linken Ecke des Bildschirmes ist die Anzahl der Reserveenten angezeigt. Wenn alle 5 Enten gestorben sind, wechselt der Bildschirm die Farbe auf rot, und SALOME erklingt. Drücken Sie jetzt den roten Aktionsknopf, um ein neues Spiel zu beginnen.

HEYNE COMPUTER-BÜCHER
Manfred S. Schmidt
DER MIKROCOMPUTER IM BERUF
Heyne-Buch Nr. 15/4, DM 9,80
Originalausgabe


```
1 REM  ***************************
2 REM  * M o t h e r - D U C K  *
3 REM  *-------------------------*
4 REM  *   TI 99/4 A            *
5 REM  * + TI Extended BASIC    *
6 REM  * + Joystick             *
7 REM  *-------------------------*
8 REM  * (c) 1984 Volker Becker *
9 REM  * Steinbacher Stra~e 10  *
10 REM * 6370 Oberursel 6       *
11 REM ***************************
12 REM
13 REM
14 REM
20 GOTO 80 :: A :: A1 :: A2 :: B1 :: B2 :: COLR :: D :: I :: K :: RUNDE :: S ::
SND :: ST :: SW :: T :: VGL :: WFARBE :: X :: Y :: Z :: ZW
30 CALL BILD :: CALL CHAR :: CALL CHARSET :: CALL CLEAR :: CALL COINC :: CALL CO
LOR :: CALL DELSPRITE :: CALL ENTE :: CALL GCHAR :: CALL GRAPH :: CALL HCHAR
40 CALL JOYST :: CALL KEY :: CALL LOCATE :: CALL MOTION :: CALL PATTERN :: CALL
POSITION :: CALL PUNKTE :: CALL SALOME :: CALL SCREEN :: CALL SOUND
50 CALL SPRITE :: CALL TITEL :: CALL WALD
80 PUNKT=0 :: ZEIT=2000 :: DUCK=4 :: WUERFEL=0 :: STATUS=0
89 OPTION BASE 1
90 DIM V(6),COL(10),FARBE(4),B(4)
91 DATA 16,5,7,13
92 !@P-
93 RESTORE 91
94 FOR I=1 TO 4 :: READ COLR :: FARBE(I)=COLR :: NEXT I
```

```
110 CALL CLEAR :: CALL CHARSET :: CALL SCREEN(4)
125 RANDOMIZE
130 CALL GRAPH
450 CALL TITEL(RUNDE,B( ))
455 CALL BILD
456 CALL CHAR(76,"03070507031F23272F2F27530202020 6C0E0A0E0C0F8C4E4F4F4E4CA404040
60")
460 CALL WALD(V( ),FARBE( ),RUNDE,COL( ))
462 CALL PUNKTE(ZEIT,PUNKT)
465 CALL SPRITE(#1,36,2,42,128)
470 CALL SALOME(B( ),1)
800 CALL CHAR(68,"00000000000000000072F3F1F0406000000000000000000307CF0C0E0E080C000
00")
810 CALL CHAR(72,"0000000000000073F0F000000000000000000000000000030FC30")
1000 CALL POSITION(#1,Z,S)
1005 ZEIT=ZEIT-2 :: CALL PUNKTE(ZEIT,PUNKT):: IF ZEIT<=0 THEN CALL ENTE(DUCK,ZW,
SW,WFARBE,I,FARBE( ),STATUS):: ZEIT=2000
1007 IF DUCK<0 THEN 9000
1010 CALL JOYST(1,X,Y):: IF X=0 THEN 1100
1020 IF X=4 THEN CALL PATTERN(#1,68)ELSE CALL PATTERN(#1,36)
1100 IF X=0 AND Y=0 THEN 1110 ELSE CALL SOUND(-30,-6,0)
1110 IF Z<58 THEN 2000 ELSE 3000
2000 IF Y=4 AND Z<38 THEN Y=0
2005 IF STATUS=1 THEN 5000
2010 CALL MOTION(#1,-Y*1.5,X*1.5)
2020 GOTO 1000
3000 IF Z>106 THEN 4000
3005 CALL COINC(ALL,ST):: IF ST THEN CALL ENTE(DUCK,ZW,SW,WFARBE,I,FARBE( ),STATU
S)
3010 CALL MOTION(#1,-Y*2,X*2):: GOTO 1000
4000 A=INT((Z+7)/8+1):: D=INT((S+7)/8+1):: IF A>24 OR D>32 OR D<1 THEN CALL ENTE
(DUCK,ZW,SW,WFARBE,I,FARBE( ),STATUS):: GOTO 1000
4020 CALL MOTION(#1,-Y*1.5,X*1.5)
4040 CALL GCHAR(A,D,VGL):: IF (VGL>135 AND VGL<140)OR(VGL>39 AND VGL<42)THEN CAL
L ENTE(DUCK,ZW,SW,WFARBE,I,FARBE( ),STATUS):: GOTO 1000
4060 CALL KEY(1,T,S):: IF T=18 THEN 4100 ELSE 1000
4100 CALL MOTION(#1,0,0)
4105 IF STATUS=1 THEN 1000
4110 FOR I=11 TO 20
4120 CALL COINC(#1,#I,12,ST)
4130 IF ST THEN 4200
4140 NEXT I
4150 GOTO 1000
4200 CALL POSITION(#I,ZW,SW):: WFARBE=COL(I-10):: CALL DELSPRITE(#I):: STATUS=1
4210 DATA 261,329,391,523,30000,523,391,329,261
4220 RESTORE 4210
4230 FOR K=1 TO 9 :: READ SND :: CALL SOUND(-250,SND,0):: NEXT K
4240 GOTO 1000
5000 WUERFEL=WUERFEL+1
5002 CALL MOTION(#1,0,0)
5005 STATUS=0
5010 CALL HCHAR(2,3,101,WUERFEL)
5012 CALL PUNKTE(ZEIT,PUNKT)
5013 FOR K=0 TO 30 :: CALL SOUND(-250,523,K,440,K):: NEXT K
5015 PUNKT=PUNKT+20*WFARBE
5020 IF WUERFEL=10 THEN WUERFEL=0 :: CALL HCHAR(2,3,128,10):: GOTO 6000
5030 GOTO 1000
6000 CALL COLOR(10,2,11,11,2,11):: CALL MOTION(#1,0,0):: CALL LOCATE(#1,4,205)
6005 RESTORE 4210
6007 FOR K=1 TO 9 :: READ SND :: CALL SOUND(-250,SND,0):: NEXT K
6010 PUNKT=PUNKT+1000 :: CALL PUNKTE(ZEIT,PUNKT)
6020 CALL KEY(1,T,S):: IF T=18 THEN 6030 ELSE 6020
```

```
6030 CALL DELSPRITE(#28):: CALL COLOR(10,11,11,11,11,11)
6040 CALL POSITION(#1,A2,B2):: CALL POSITION(#2,A1,B1):: A=INT((A2+7)/8+1):: D=I
NT((B2+7)/8+1)
6050 IF A<1 OR A>8 OR D<1 OR D>32 THEN 7000
6060 CALL GCHAR(A,D,VGL):: IF VGL=122 OR VGL=123 THEN 7500
6070 CALL MOTION(#2,(A2-A1)/4,(B2-B1)/4)
6080 CALL JOYST(1,X,Y):: CALL MOTION(#1,-Y*1.5,X*1.5):: CALL SOUND(-30,-6,0):: I
F X=4 THEN CALL PATTERN(#1,68)ELSE IF X=-4 THEN CALL PATTERN(#1,36)
6090 CALL COINC(#1,#2,10,ST):: IF ST THEN 7000
6100 ZEIT=ZEIT-2 :: CALL PUNKTE(ZEIT,PUNKT):: IF ZEIT<=0 THEN 7000 ELSE 6040
7000 CALL MOTION(#1,0,0,#2,0,0):: CALL ENTE(DUCK,ZW,SW,WFARBE,I,FARBE( ),0)
7005 IF DUCK<0 THEN 9000
7010 CALL HCHAR(16,1,32,256):: GOTO 460
7500 CALL MOTION(#1,0,0,#2,0,0):: CALL DELSPRITE(#2)
7505 PUNKT=PUNKT+ZEIT :: ZEIT=0 :: CALL PUNKTE(ZEIT,PUNKT):: ZEIT=2000
7507 FOR K=0 TO 30 :: CALL SOUND(-250,523,K,440,K):: NEXT K
7510 RUNDE=RUNDE+1 :: IF RUNDE>10 THEN RUNDE=10
7520 CALL HCHAR(16,1,32,256)
7530 GOTO 460
8000 CALL ENTE(DUCK,ZW,SW,WFARBE,I,FARBE( ),STATUS)
8010 GOTO 1000
9000 CALL DELSPRITE(ALL):: CALL SCREEN(7):: CALL SALOME(B( ),1)
9010 GOTO 80
27999 !@P+
28000 SUB TADEL
28001 GOTO 28010 :: A :: L :: I :: CALL SOUND :: !@P-
28010 DATA 750,195,250,220,500,233,500,195,1000,277
28020 RESTORE 28010
28030 FOR I=1 TO 5 :: READ L,A :: CALL SOUND(L,A,0):: NEXT I
28035 !@P+
28040 SUBEND
28050 SUB PUNKTE(ZEIT,PUNKT)
28051 !@P-
28060 DISPLAY AT(24,1):USING "  P : ######    Z : ####":PUNKT,ZEIT
28070 !@P+
28080 SUBEND
28200 SUB ENTE(DUCK,ZW,SW,WFARBE,I,FARBE( ),STATUS)
28201 GOTO 28210 :: CALL TADEL :: !@P-
28210 CALL MOTION(#1,0,0)
28220 CALL PATTERN(#1,72)
28230 CALL TADEL
28240 DUCK=DUCK-1
28245 CALL HCHAR(1,3,128,4)
28250 IF DUCK<=0 THEN 28265
28260 CALL HCHAR(1,3,121,DUCK)
28265 CALL SPRITE(#1,36,2,42,128)
28267 IF DUCK<0 THEN CALL DELSPRITE(#1)
28268 IF STATUS=0 THEN SUBEXIT
28269 CALL SPRITE(#I,96,FARBE(WFARBE),ZW,SW,0,0):: STATUS=0
28270 !@P+
28271 SUBEND
29000 SUB TITEL(RUNDE,B( ))
29001 GOTO 29010 :: Z$ :: I :: P :: CALL MAGNIFY :: CALL VCHAR :: !@P-
29010 CALL BILD
29020 DATA 98,D,114,U,130,C,146,K
29035 CALL MAGNIFY(2)
29040 DISPLAY AT(9,9)SIZE(12):"   MOTHER   " :: DISPLAY AT(14,9)SIZE(12):"
      "
29050 CALL VCHAR(10,11,32,4):: CALL VCHAR(10,22,32,4)
29053 DISPLAY AT(24,3):"@ 1984 BY VOLKER BECKER"
29055 RESTORE 29020
29060 FOR I=1 TO 4 :: READ P,Z$ :: CALL SPRITE(#I,ASC(Z$),11,81,P):: NEXT I
```

```
29070 DISPLAY AT(18,1):"BEGINN BEI RUNDE <1-10> :" :: CALL SALOME(B( ),5)
29080 ACCEPT AT(18,27)SIZE(2)VALIDATE(DIGIT):RUNDE
29090 IF RUNDE<1 THEN RUNDE=1
29100 IF RUNDE>10 THEN RUNDE=10
29110 CALL HCHAR(16,1,32,288):: CALL DELSPRITE(ALL)
29115 !@P+
29120 SUBEND
30000 SUB SALOME(B( ),KY)
30001 GOTO 30010 :: AZ :: I :: J :: K :: M1 :: M2 :: S :: STV :: T :: !@P-
30010 DATA 2,220,261,349,440,6,261,440,2,293,466
30020 DATA 4,195,233,293,391,6,233,391,1,349,440,1,391,466,1,329,391,2,349,440,1
,391,466,2,466,587,2,391,466
30040 DATA 2,220,261,349,440,8,440,698
30045 DATA 2,220,261,329,440,6,523,659,2,329,523
30050 DATA 1,195,233,293,391,4,391,466
30055 DATA 1,233,261,329,440,2,391,466,1,329,523,1,261,440,1,195,233,293,391,1,3
29,391,2,391,440,1,329,391
30060 DATA 1,195,233,293,391,2,233,293,2,261,329
30065 DATA 2,220,261,349,440,7,261,440,1,30000,30000
30070 DATA 2,220,261,349,440,6,261,440,2,293,466
30075 DATA 4,195,233,293,391,6,233,391,1,349,440,1,391,466,1,329,391,2,349,440,1
,391,466,2,466,587,2,391,466
30085 DATA 2,220,261,349,440,7,440,698,1,30000,30000
30087 DATA 4,220,261,349,440,6,440,698,2,391,659,6,329,523,1,391,466,1,440,523
30091 DATA 2,195,233,293,391,4,440,554,2,440,523,2,261,440
30093 DATA 2,220,261,349,440,8,220,349,0
30100 RESTORE 30010
30105 AZ,STV=0
30110 READ K :: IF K=0 THEN 30100
30115 READ B(1),B(2),B(3),B(4)
30120 FOR I=1 TO K
30130 FOR J=1 TO 4
30135 STV=STV+1 :: IF STV>AZ THEN READ AZ,M1,M2 :: STV=1
30140 CALL SOUND(400,M1,5,M2,4,B(J),9):: CALL KEY(KY,T,S):: IF S<>0 THEN SUBEXIT
30150 NEXT J :: NEXT I
30160 GOTO 30110
30170 !@P+
30200 SUBEND
31000 SUB BILD
31100 GOTO 31165 :: CALL PYRAMIDE :: !@P-
31165 CALL COLOR(13,1,11):: CALL HCHAR(1,1,128,224)
31170 CALL COLOR(10,11,11,11,11,11,14,13,1,2,7,1):: CALL PYRAMIDE
31180 CALL COLOR(10,2,11,11,2,11)
31190 CALL COLOR(9,5,11,12,7,11):: CALL HCHAR(9,1,100,192):: CALL HCHAR(8,1,120,
32):: CALL HCHAR(15,1,120,32)
31195 CALL HCHAR(1,3,121,4)
31197 CALL HCHAR(7,5,122):: CALL HCHAR(7,6,123)
31199 !@P+
31200 SUBEND
31210 SUB WALD(V( ),FARBE( ),RUNDE,COL( ))
31211 GOTO 31212 :: C :: I :: K :: SP :: Z :: !@P-
31212 CALL MAGNIFY(3)
31215 FOR I=1 TO RUNDE+2
31220 Z=RND*5+16 :: SP=RND*30+1 :: CALL GCHAR(Z,SP,V(1)):: CALL GCHAR(Z+1,SP,V(2
)):: CALL GCHAR(Z,SP+1,V(3)):: CALL GCHAR(Z+1,SP+1,V(4))
31222 CALL GCHAR(Z+2,SP,V(5)):: CALL GCHAR(Z+2,SP+1,V(6))
31223 FOR K=1 TO 6 :: IF V(K)<>32 THEN 31220
31224 NEXT K
31225 CALL HCHAR(Z,SP,136):: CALL HCHAR(Z+1,SP,137):: CALL HCHAR(Z,SP+1,138):: C
ALL HCHAR(Z+1,SP+1,139)
31230 CALL HCHAR(Z+2,SP,40):: CALL HCHAR(Z+2,SP+1,41):: NEXT I
31315 FOR I=11 TO 20
```

```
31320 Z=INT(RND*6+16):: SP=INT(RND*29+1):: CALL GCHAR(Z,SP,V(1)):: CALL GCHAR(Z+
1,SP,V(2)):: CALL GCHAR(Z,SP+1,V(3)):: CALL GCHAR(Z+1,SP+1,V(4))
31322 CALL GCHAR(Z+2,SP,V(5)):: CALL GCHAR(Z+2,SP+1,V(6))
31323 FOR K=1 TO 6 :: IF V(K)<>32 THEN 31320
31324 NEXT K
31325 C=INT(RND*3+1):: COL(I-10)=C :: CALL SPRITE(#I,96,FARBE(C),Z*8-7,SP*8-7)
31327 CALL HCHAR(Z,SP,33,2):: CALL HCHAR(Z+1,SP,33,2)
31330 NEXT I
31340 CALL SPRITE(#28,124,7,1,205)
31345 CALL COLOR(10,11,11,11,11,11)
31350 CALL SPRITE(#3,140,11,65,240):: CALL SPRITE(#4,132,11,81,5):: CALL SPRITE(
#5,140,11,97,240)
31360 IF RUNDE>1 THEN CALL SPRITE(#6,132,11,81,70)
31370 IF RUNDE>2 THEN CALL SPRITE(#7,140,11,65,180)
31380 IF RUNDE>3 THEN CALL SPRITE(#8,140,11,97,170)
31390 IF RUNDE>5 THEN CALL SPRITE(#9,132,11,81,175)
31400 IF RUNDE>7 THEN CALL SPRITE(#21,140,11,97,40)
31410 IF RUNDE>=10 THEN CALL SPRITE(#22,140,11,65,30)
31420 CALL MOTION(#3,0,-(8+RUNDE),#7,0,-(8+RUNDE),#22,0,-(8+RUNDE))
31430 CALL MOTION(#4,0,4+RUNDE,#6,0,4+RUNDE,#9,0,4+RUNDE)
31440 CALL MOTION(#5,0,-(1+RUNDE),#8,0,-(1+RUNDE),#21,0,-(1+RUNDE))
31445 CALL SPRITE(#2,76,14,193,8,0,0)
31447 !@P+
31450 SUBEND
32000 SUB PYRAMIDE
32001 !@P-
32010 DISPLAY AT(3,24)SIZE(2):"hi" :: DISPLAY AT(4,24)SIZE(3):"jkl" :: DISPLAY A
T(5,23)SIZE(4):"mnno" :: DISPLAY AT(6,23)SIZE(5):"pqqrs"
32020 DISPLAY AT(7,23)SIZE(5):"ttttu"
32025 !@P+
32030 SUBEND
32040 SUB GRAPH
32045 !@P-
32080 CALL CHAR(100,RPT$("F",16)):: CALL CHAR(120,"66FFF9FD9FBFFF99")
32097 CALL CHAR(33,RPT$("0",16))
32100 CALL CHAR(96,"0000000102040800C0A09080804020100000080402010081828488888890A0
C080")
32102 CALL CHAR(136,"0103070F1F03070F1F3F7F0F1F3F7FFF80C0E0F0F8C0E0F0F8FCFEF0F8F
CFEFF"):: CALL CHAR(40,RPT$("03",8)):: CALL CHAR(41,RPT$("C0",8))
32105 CALL CHAR(128,RPT$("0",16))
32130 CALL CHAR(104,"00010204080C0A09804020100818284808081422418 0C1A2888894A2C18
0C1A2")
32135 CALL CHAR(124,"00000000001F3F7F7F7F7F7F3F3F1F0000000608102040 80E0F0F0F0F0E
0C000")
32140 CALL CHAR(108,"0000000000808080000000010204080C948888492A1C081C80808040201
00818")
32150 CALL CHAR(112,"0A09080814224180 2A49888894A2C1802848888894A2C1800000000000
00080")
32155 CALL CHAR(140,"0000000000000003FDFF0703000000000000000002060E1F3FFFEFFF3C18
00000")
32156 CALL CHAR(132,"0000000040687CFFF7FFFCF8302000000000000000000C0BFFFE0C0000
00000")
32160 CALL CHAR(116,"C1A2948888492A1C808080808000000008")
32170 CALL CHAR(36,"0000000000000C3E0F0307070103000000000000000000E0F4FCF82060
0000")
32180 CALL CHAR(121,"00060FBCFC101800")
32190 CALL CHAR(101,"1028446C54543810")
32200 CALL CHAR(122,"617942FF341B0E078CEF10FFBC5870E0")
32210 CALL CHAR(64,"3C4299A1A199423C")
32220 !@P+
32300 SUBEND
```

Preiswert drucken, aber?

Um mit dem TI-99-4A im DIN-A 4-Format zu drucken, war bisher eine aufwendige, kostenintensive Peripherie erforderlich. Nun aber geht es einfacher und preiswerter.

Es gibt verschiedene Interfaces, die direkt an die Konsole angeschlossen werden können, und mit diesen kann dann ein Drucker direkt verbunden werden. Für diesen Bericht wurden getestet:
Das Interface MBI der Fa. Klein.
Der Drucker stx-80 der Fa. star.
Die Montage beider Geräte dauert keine 5 Minuten.
Bevor wir die Arbeitsweise beschreiben, sollen zwei „Nachteile" des Drukkers nicht verschwiegen werden:
1. Es wird Thermopapier benutzt.
2. Die Bedienungsanleitung ist dürftig und nur in englisch.

Zu 1 muß man aber bedenken, daß Farbbänder für Normaldruck auch nicht eben billig sind und Reparaturen am Thermodrucker wegen der nichtvorhandenen komplizierten Mechanik, z. B. des Druckkopfes, seltener und preiswerter sein dürften.
Zu 2. Soll dieser Bericht Abhilfe schaffen. Wir wollen nicht zuviel technische Daten bringen, sondern praktische Hilfe geben, die aus der Anleitung nicht oder nur schwer zu entnehmen ist.
Was leistet der stx-80 nun? – Da wären zu erwähnen:
Sehr leiser Druck mit 60 Zeichen/sec.
80 Zeichen pro Zeile, in Breitschrift 40.
Deutsche Umlaute Ä, Ö, Ü und ß.
Groß- und Kleinschrift (siehe Bild 1).
Breitdruck (siehe Bild 2).
64 vordefinierte Grafikzeichen (siehe Bild 3 u. 4).
Bit Image Mode (Einzelpunktansteuerung).

Änderung des Zeilenabstandes (wichtig z. B. bei Balkendiagrammen)

Alle diese Eigenschaften können nach Einstellung der DIP-Schalter softwaremäßig gesteuert werden. Genau damit wollen wir uns befassen. Denn hier liegt die Schwäche des Manuals.
Das beste Schriftbild ergibt sich, wenn man in Großbuchstaben schreibt. Daher braucht man auch nur über die eckige Klammer (auf) ein Ä, über dem Schrägstrich links unten ein Ö (Z-Taste) und über der eckigen Klammer (zu) ein Ü zu kleben.
Angesprochen wird der Drucker mit „RS232.BA=4800" ohne Zwischenraum! Die 1. PGM.-Zeile heißt dann:
100 open #1:„RS232.BA=4800", output,
und gedruckt wird mit:
110 Print#1: „Testtext".
Will man dieses Wort in Breitschrift drucken, so heißt der Befehl: CHR$(14); also:
110 Print#1:CHR$(14); „Testext".
Der Befehl CHR$(14) muß vor jeder Zeile stehen, die dann die Hälfte nämlich statt 80 Zeichen hat. Soll hinter der Breitschrift wieder in Normalschrift gedruckt werden, muß der Befehl mit CHR
CHR#(20); wieder aufgehoben werden. Also:
110 Print#1:CHR$(14);„Testext"; CHR$(20); „Testext".
Zu beachten ist das Zusammentreffen von Grafik und Breitschrift. Hier müssen die Befehle in aufsteigender Reihenfolge eingegeben werden. Grafik=(15):
110 Print#1:CHR$(14); CHR$(15); „Testext".

Und hier nun die Liste der Steuerbefehle:

CHR$(9); = Druck ab Spalte 11
CHR$(10); = 2 Leerzeilen
CHR$(12); = Vorschub auf die nächste Seite
CHR$(13); = 1 Leerzeile
CHR$(14); = Breitschriftmodus
CHR$(15); = Grafikmodus
CHR$(18); = Aufhebung des Grafikmodus
CHR$(20); = Aufhebung des Breitschriftmodus
CHR$(27);„A"CHR$(1); = Zeilenabstand 1/11 Zoll
CHR$(27);„A"CHR$(2); = Zeilenabstand 1/6 Zoll

1/11 Zoll bedeutet praktisch keinen Abstand zwischen den Zeilen und wird z. B. bei den Grafiken benutzt. 1/6 Zoll ist die Rückstellung in den Normalabstand.
Achten Sie darauf, den Grafikmodus immer aufzuheben, da der Drucker sonst alles folgende ebenfalls im Grafikmodus ausgibt (nicht zu verwechseln mit dem Bit Image Modus).
Eine Tabelle der Grafikzeichen und welches Tastaturzeichen sie anschlagen müssen, zeigen die Bilder 3 und 4. Bild 4 zeigt die Wiedergabe durch den Drucker. Bei Bild 3 helfen die Umrandungen, die Zeichen, z. B. senkrechte Striche, richtig zu plazieren.
Um beispielsweise Balkendiagramme auszudrucken, darf praktisch kein Abstand zwischen den Zeilen bestehen. Dazu dient der vorletzte Befehl aus der vorstehenden Liste. Wird hierbei mit For-Next-Schleifen gearbeitet, muß der Befehl
CHRY$(27);„A";CHR$(1);
vor der Schleife stehen und der ihn aufhebende Befehl dahinter.
Die Stellung der DIP-Schalter, die leicht zugänglich unter Papierrolle liegen, ist für die deutsche Schreibweise:
1, 3 und 5 nach hinten (ON),
2, 4 und 6 nach vorne (OFF.
Damit kennen Sie die wichtigsten Eigenschaften und können nun Ihren TI noch vielseitiger einsetzen.
Zum Schluß noch ein Lob der Fa. star, die dem Autor, der sich als Privatmann meldete, bei einigen der obigen Probleme mit telefonischen Rückrufen bei der Lösung geholfen hat. Da schöpft man, weil ungewöhnlich in der Branche, gleich wieder etwas Hoffnung.

Herbert May
star Europe GmbH
Frankfurter Allee 1–3
6236 Eschborn-TS
Klein-Computer
Münzstraße 34 A
5600 Wuppertal 2

Neue Produktlinie Zubehör für alle PC-Computer-Systeme unter der Marke ASCOM

Zur Hifi '84 hat die DYNAMICS marketing GmbH ihr Angebot ganz wesentlich erweitert. Unter der Marke ASCOM bringt DYNAMICS ein umfangreiches Computer-Zubehör-Programm auch für den PC-Bereich auf den Markt.
Die einzelnen Produktgruppen sind
1. Kabel
2. Buchsen, Stecker, Kupplungen
3. Interface, Buffer, Weiche
4. Akustik-Koppler, Lightpen
5. Computer-Pflege-Serie

1. Kabel

Zum Verlängern, Anschließen und Verbinden des Computers mit Peripheriegeräten sind gute und passende Kabel unerläßlich, deshalb bietet Dynamics unter der Marke ASCOM folgende Kabel an:
Computer-Rundkabel, 9adrig und Computer-Rundkabel 25adrig, beide mit Abschirmung gegen Störungen z. B. von benachbarten Geräten, 2 m Länge.
Computer-Flachband-Kabel, 25adrig, flache Kabel sparen Platz, sind mechanisch flexibler und widerstehen besser äußerem Druck.

Topprogramm

Ausgezeichnet von der Redaktion als Top-Programm in „Computronic". Bewertet wurden Spielidee, Bewegungsablauf und Grafik.

Sie steigen in den Alltag einer Flaschenfabrik ein. Dort müssen Sie Bierkisten zusammenstellen. Die Arbeit wird Ihnen aber durch zahlreiche Hindernisse erschwert.

Nach dem Start mit „RUN" erklingt eine Melodie. Danach werden die Bildnummer und die Schwierigkeitsphase angezeigt. Eine Phase besteht aus vier Bildern (4 Aufgaben). Nach der Anzeige, die nach jedem geschafften Bild eingeblendet wird, erscheint das Bild Nr. 1. Oben links wird die Anzahl der verfügbaren Leben angezeigt, d. h., am Anfang hat jeder Spieler drei Leben. Links unten erscheint das erste Männchen, welches mit dem Joystick gesteuert wird. Der Bildschirmaufbau besteht aus mehreren Etagen, die durch Leitern verbunden sind. Auf den Etagen befinden sich Fließbänder, die Flaschen transportieren. Die Anzahl der Fließbänder nimmt bis zum vierten Bild zu, und ab Bild Nr. 2 werden diese teilweise unsichtbar. Außerdem befinden sich 9 Bierfässerteilchen auf dem Bildschirm, die für den Spieler sehr wichtig sind, denn sie haben beim Start eines Bildes ihre bestimmte Anfangsposition, die bei den vier Bildern jeweils unterschiedlich ist. Durch Berühren eines Fässerteilchens mit dem Männchen, verändert sich die Position dieses Teilchens und die direkt unter ihm liegenden Teilchen. Wie und was durch so eine Berührung passiert, ist aus den Beispielen 1 und 2 zu entnehmen. Hat man alle Bierfässerteile (durch Berührung) nach unten transportiert, so erkennt man 3 Bierfässer. Wenn man dieses erreicht hat, ist das Bild bzw. die Aufgabe erfüllt, und es erscheint die Anzeige der Phase und Bildschirmnummer. Danach folgt das nächste Bild. Dieses hört sich alles relativ einfach an, doch es gibt einen Schwachpunkt des Männchens. Es verträgt keinen Alkohol. Deswegen darf es keine auf dem Bildschirm befindlichen Flaschen berühren, da sie Alkohol enthalten. Dieses kann durch einen Sprung verhindert werden (die Steuerung des Männchens ist aus dem Beispiel 3 zu entnehmen). Eine weitere Gefahr ist die Zeit. Oben am Bildschirm erkennt man eine Etage, auf dem das Männchen des Spielers symbolisiert wird. Von rechts her nähert sich eine Whiskyflasche. Sollte diese zu dem symbolisierten Männchen gelangen, so hat der Spieler ein Leben verloren. Da dieses sehr schnell geht, hat der Spieler noch eine Chance, durch Berühren des Malzbieres, die Flasche wieder von vorn (also von rechts) starten zu lassen. Das Malzbier erscheint an einer bestimmten Position. Wann es erscheint, hängt von dem Berühren eines der drei Oberteile der Bierfässer ab. Hierzu ein Beispiel: das Malzbier ist nicht auf dem Bild, aber wenn man jetzt eines der Oberteile berührt, dann erscheint es. Bei nochmaliger Berührung verschwindet es und bei der nächsten Berührung erscheint es wieder. Also ein bestimmter Rhythmus, das Malzbier muß eindeutig berührt werden.

Punkte werden beim Versetzen eines Faßteiles gegeben oder nach Beendigung eines Bildes. Die Entfernung der Whiskyflasche zum Männchen ist für die Höhe der Punkte ausschlaggebend. Die aktuelle Punktzahl wird nach Verlust eines Männchens oder bei anderen Unterbrechungen angezeigt, aber nicht während des Spieles. Sollte man eine Phase geschafft haben, so erhält man ein zusätzliches Leben. Sollte man alle Leben verbraucht bzw. verspielt haben, so erscheint „GAME-OVER".

Durch Betätigung einer Taste, beginnt das Spiel von vorn. Ansonsten ist das Programm musikalisch untermalt.

```
10 CALL SCREEN(2):: FOR I=3 TO 8 :: CALL COLOR(I,9,1):: NEXT I
20 CALL CLEAR
30 DISPLAY AT(9,9):"ALKOHOLVERBOT" :: DISPLAY AT(11,14):"VON" :: DISPLAY AT(13,1
0):"JENS BARTHE"
40 CALL MUSIK
50 CALL KEY(1,K,S):: IF K=18 THEN 60 ELSE CALL KEY(0,K,S):: IF S=0 THEN 50
60 CALL CLEAR :: CALL MAGNIFY(3)
70 CALL CHAR(95,"1C3E1C083E1C1C14",97,"0103000700070F0F",98,"FFFF00FF00FFFFFF",9
```

```
9,"80C000E000E0F0F0")
80 CALL CHAR(100,"0F0F07070003000 1",101,"FFFFFFFF00FF00FF",102,"F0F0E0E000C0008"
,113,"FFFFFFFF")
90 CALL CHAR(104,"0F0F080A080A080F",105,"FFFFA2AEA2AEA2FF",106,"F0F030B07070B0F"
,113,"FFFFFFFF")
100 CALL CHAR(120,"192640405F3F5F5B",121,"B8440204F8F0FEBF",122,"395A5B9B7B5B1F1
F",123,"33B3B3BEBCB0F0F0",124,"FF81FF81FF81FF81")
110 CALL CHAR(36,"01030F0207030101 0B0703030101010380C0C0E0E0C0800080808080000000
00")
120 CALL CHAR(40,"0103030707030301030303030101010180C0F040E0C00000A0C0808000000
80")
130 CALL CHAR(60,"03070F0F0F0713130F0707070404040C0080C0C0C0802020C0808080808080
C0")
140 CALL CHAR(108,"0302030101010101060830 4D4D26100FC040C0808080808060100CB2B2640
8F0")
150 CALL CHAR(116,"01030F0207030101 0B0703030306081880C0C0E0E0C08000C0A0808080 80E
020")
160 CALL CHAR(128,"0103030707030301030 70B0303020E0880C0F040E0C00000A0C0808080C02
030")
170 CALL CHAR(132,"0000000000000000000444444EEAAAAEE0000000000000000000444444EEAAA
AEE")
180 CALL CHAR(136,"0003070302020202020 40505080908070 0C0E0C0404040404020A02090901
0E0")
190 CALL CHAR(140,"0003010101010101010 204090908040300E0C0C0404040404020D00808C81
0E0")
200 IF SC>HS THEN HS=SC
210 M=2 :: BI,SC=0 :: PH=1
220 BI=BI+1 :: IF BI>4 THEN 230 ELSE 240
230 M=M+1 :: PH=PH+1 :: BI=1
240 CALL CLEAR :: CALL DELSPRITE(ALL):: DISPLAY AT(11,11):"BILD :";BI :: DISPLAY
 AT(13,10):"PHASE :";PH :: FOR I=1 TO 300 :: NEXT I
250 Q1=INT(-2-PH/2):: Q2=INT(4+PH/3):: Q3=INT(-7-PH):: Q4=INT(-3-PH/2):: Q5=INT(
5+PH/3):: Q6=INT(3+PH/3):: Q7=INT(2+PH/3)
260 CALL CLEAR :: ON BI GOSUB 320,410,540,670
270 DISPLAY AT(24,1):"HI-SCORE:";HS :: DISPLAY AT(24,17):"SCORE:";SC
280 CALL HCHAR(2,2,95,M)
290 CALL SPRITE(#20,128,16,1,40,#21,108,13,1,256,0,-1):: CALL SPRITE(#1,40,16,15
3,6):: CALL HCHAR(3,6,113,27)
300 GOTO 810
310 ! BILD 1
320 CALL COLOR(9,11,1,10,11,1,11,16,1,12,5,1)
330 CALL HCHAR(1,1,32,8):: A(1),B(1),C(1)=8 :: A(2),B(2),C(2)=13 :: A(3),B(3),C(
3)=18 :: AA=7 :: BB=17 :: CC=27
340 CALL HCHAR(9,1,113,32):: CALL HCHAR(14,1,113,32):: CALL HCHAR(19,1,113,32)::
 CALL HCHAR(22,1,113,2)
350 CALL VCHAR(9,2,124,5):: CALL VCHAR(14,12,124,5):: CALL VCHAR(14,22,124,5)::
CALL VCHAR(9,31,124,5)
360 CALL VCHAR(14,3,124,5):: CALL VCHAR(19,2,124,3):: CALL VCHAR(14,32,124,5)
370 CALL SPRITE(#2,140,14,129,60,#3,140,14,129,188,#4,136,4,89,80,#5,136,4,89,20
8)
380 CALL SPRITE(#6,132,7,49,40,#7,132,7,49,168)
390 CALL MOTION(#2,0,Q1,#3,0,Q1):: CALL MOTION(#4,0,Q2,#5,0,Q2):: CALL MOTION(#6
,0,Q3,#7,0,Q3):: RETURN
400 ! BILD 2
410 CALL COLOR(11,8,1,12,5,1)
420 A(1)=6 :: C(1)=5 :: A(2)=11 :: C(2)=10 :: A(3)=16 :: C(3)=15
430 CALL HCHAR(1,1,32,8):: CALL HCHAR(7,2,113,9):: CALL HCHAR(6,14,113,7):: CALL
 HCHAR(6,26,113,7):: CALL HCHAR(8,20,113,5)
440 CALL HCHAR(9,16,113,4):: CALL HCHAR(11,11,113,4):: CALL HCHAR(11,24,113,9)::
 CALL HCHAR(12,4,113,6)
450 CALL HCHAR(14,1,113,22):: CALL HCHAR(16,25,113,4):: CALL HCHAR(16,31,113,2):
: CALL HCHAR(17,6,113,4)
460 CALL HCHAR(19,2,113,5):: CALL HCHAR(19,9,113,23):: CALL HCHAR(22,1,113,3)::
```

```
CALL HCHAR(4,2,113,3)
470 CALL VCHAR(4,3,124,3):: CALL VCHAR(7,2,124,12):: CALL VCHAR(7,4,124,5):: CAL
L VCHAR(7,11,124,4):: CALL VCHAR(6,14,124,5)
480 CALL VCHAR(6,19,124,3):: CALL VCHAR(6,20,124,2):: CALL VCHAR(8,24,124,3):: C
ALL VCHAR(12,9,124,2):: CALL VCHAR(11,32,124,5)
490 CALL VCHAR(14,21,124,5):: CALL VCHAR(16,31,124,3):: CALL VCHAR(17,6,124,2)::
 CALL VCHAR(17,9,124,2):: CALL VCHAR(19,3,124,3)
500 CALL VCHAR(6,31,124,5):: CALL SPRITE(#2,140,14,129,60,#3,140,14,129,188,#4,1
36,4,89,80,#5,136,4,89,208)
510 CALL SPRITE(#6,132,7,33,70,#7,132,7,65,198):: CALL MOTION(#6,0,PH+5,#7,0,Q3)
520 CALL MOTION(#2,0,Q4,#3,0,Q4,#4,0,Q5,#5,0,Q5):: RETURN
530 ! BILD 3
540 CALL COLOR(11,5,1,12,14,1)
550 C(1)=6 :: C(2)=11 :: C(3)=16
560 CALL HCHAR(1,1,32,8):: CALL HCHAR(7,1,113,9):: CALL HCHAR(7,13,113):: CALL H
CHAR(6,21,113):: CALL HCHAR(8,22,113)
570 CALL HCHAR(9,16,113,5):: CALL HCHAR(12,4,113,6):: CALL HCHAR(12,14,113):: CA
LL HCHAR(14,15,113,7):: CALL HCHAR(12,26,113,3)
580 CALL HCHAR(17,1,113,8):: CALL HCHAR(17,26,113,7):: CALL HCHAR(19,9,113,4)::
CALL HCHAR(19,16,113,3):: CALL HCHAR(19,22,113,4)
590 CALL HCHAR(14,25,113):: CALL HCHAR(22,1,113,3):: CALL VCHAR(7,2,124,10):: CA
LL VCHAR(17,3,124,5):: CALL VCHAR(12,4,124,5)
600 CALL VCHAR(7,9,124,5):: CALL VCHAR(17,9,124,2):: CALL VCHAR(7,14,124,5):: CA
LL VCHAR(12,15,124,2):: CALL VCHAR(6,20,124,3)
610 CALL VCHAR(9,21,124,3):: CALL VCHAR(6,22,124,2):: CALL VCHAR(12,25,124,2)::
CALL VCHAR(17,25,124,2):: CALL VCHAR(12,29,124,5)
620 CALL HCHAR(7,26,113,7):: CALL HCHAR(12,21,113,2):: CALL VCHAR(7,31,124,10)
630 CALL SPRITE(#2,140,15,129,60,#3,140,15,113,188,#4,136,4,73,80,#5,132,7,33,19
8)
640 CALL SPRITE(#6,136,4,89,208,#7,132,7,49,70)
650 CALL MOTION(#2,0,Q1,#3,0,Q6,#4,0,Q2,#5,0,Q7):: CALL MOTION(#6,0,Q1,#7,0,Q4)::
 RETURN
660 ! BILD 4
670 CALL COLOR(11,13,1,12,7,1)
680 A(1)=7 :: B(1)=5 :: C(1)=4 :: A(2)=12 :: B(2)=10 :: C(2)=9 :: A(3)=17 :: B(3
)=15 :: C(3)=14 :: CC=26
690 CALL HCHAR(1,1,32,8):: CALL HCHAR(5,24,113,4):: CALL HCHAR(5,32,113):: CALL
HCHAR(6,16,113,3):: CALL HCHAR(8,4,113,12)
700 CALL HCHAR(8,19,113):: CALL HCHAR(8,22,113,2):: CALL HCHAR(10,24,113,4):: CA
LL HCHAR(10,31,113):: CALL HCHAR(11,12,113)
710 CALL HCHAR(13,4,113,7):: CALL HCHAR(14,18,113,6):: CALL HCHAR(15,11,113):: C
ALL HCHAR(15,25,113,3):: CALL HCHAR(16,17,113)
720 CALL HCHAR(18,1,113,3):: CALL HCHAR(18,6,113,3):: CALL HCHAR(18,12,113):: CA
LL HCHAR(18,16,113,3):: CALL HCHAR(18,22,113,3)
730 CALL HCHAR(18,28,113,5):: CALL HCHAR(20,1,113,2):: CALL HCHAR(22,1,113,3)::
CALL VCHAR(18,2,124,2):: CALL VCHAR(20,3,124,2)
740 CALL VCHAR(8,3,124,10):: CALL VCHAR(11,10,124,2):: CALL VCHAR(11,11,124,4)::
 CALL VCHAR(8,12,124,3):: CALL VCHAR(15,12,124,3)
750 CALL VCHAR(6,15,124,2):: CALL VCHAR(16,16,124,2):: CALL VCHAR(16,18,124,2)::
 CALL VCHAR(6,19,124,2):: CALL VCHAR(11,20,124,3)
760 CALL VCHAR(5,23,124,3):: CALL VCHAR(10,23,124,4):: CALL VCHAR(8,24,124,2)::
CALL VCHAR(15,24,124,3):: CALL VCHAR(15,28,124,3)
770 CALL VCHAR(5,31,124,5):: CALL VCHAR(10,32,124,8):: CALL HCHAR(11,16,113,4)
780 CALL SPRITE(#2,140,14,121,60,#3,140,14,97,188,#4,136,4,81,80,#5,136,4,65,198
)
790 CALL SPRITE(#6,132,7,41,70,#7,132,7,17,208)
800 CALL MOTION(#2,0,Q1,#3,0,Q6,#4,0,Q2,#5,0,Q2):: CALL MOTION(#6,0,Q1,#7,0,Q4)::
 RETURN
810 ON ERROR 1480
820 ! AUFSTELLUNG DER FRESSERTEILE
830 FOR I=1 TO 3 :: IF I=1 THEN 840 ELSE IF I=2 THEN 850 ELSE 860
840 P1=A(1):: P2=A(2):: P3=A(3):: P4=AA :: GOTO 870
850 P1=B(1):: P2=B(2):: P3=B(3):: P4=BB :: GOTO 870
```

```
860 P1=C(1):: P2=C(2):: P3=C(3):: P4=CC :: GOTO 870
870 DISPLAY AT(P1,P4-3)SIZE(3):"abc" :: DISPLAY AT(P2,P4-3)SIZE(3):"hij" :: DISP
LAY AT(P3,P4-3)SIZE(3):"def" :: NEXT I
880 ! STEUERRUNG
890 GOSUB 1050 :: CALL POSITION(#1,Y,X):: CALL JOYST(1,XX,YY):: CALL KEY(1,K,S)
900 IF YY<>0 AND XX<>0 THEN X2=0
910 IF K<>-1 THEN X1=XX :: IF XX=4 THEN 1010 ELSE IF XX=-4 THEN 1030 ELSE 1000
920 IF YY<>0 OR XX<>0 THEN 950
930 IF X1=-4 THEN CALL PATTERN(#1,36)ELSE CALL PATTERN(#1,40)
940 GOTO 890
950 GOSUB 1050 :: IF XX=-4 THEN X=X-8 :: X1=XX :: CALL PATTERN(#1,116)ELSE IF XX
=4 THEN X=X+8 :: X1=XX :: CALL PATTERN(#1,128)ELSE 970
960 GOTO 1060
970 IF YY=4 THEN 1150 ELSE IF YY=-4 THEN 1160
980 GOTO 1060
990 ! SPRUENGE
1000 FOR I=14 TO -14 STEP -2 :: CALL MOTION(#1,-I,0):: FOR II=1 TO 8 :: NEXT II
:: NEXT I :: CALL MOTION(#1,0,0):: GOTO 1060
1010 CALL PATTERN(#1,128):: FOR I=12 TO -12 STEP -2 :: CALL MOTION(#1,-I,5):: GO
SUB 1050 :: FOR II=1 TO 8 :: NEXT II :: NEXT I :: X=X+32
1020 CALL MOTION(#1,0,0):: GOTO 1060
1030 CALL PATTERN(#1,116):: FOR I=12 TO -12 STEP -2 :: CALL MOTION(#1,-I,-5):: G
OSUB 1050 :: FOR II=1 TO 8 :: NEXT II :: NEXT I :: X=X-32
1040 CALL MOTION(#1,0,0):: GOTO 1060
1050 CALL COINC(ALL,CF):: IF CF=-1 THEN 1480 :: RETURN
1060 AE=INT((X+10)/8):: IF AE>32 THEN X=X-8 ELSE IF AE<2 THEN X=X+8
1070 GOSUB 1050 :: CALL LOCATE(#1,Y,X):: Y=INT((Y+15)/8):: X=INT((X+10)/8):: CAL
L GCHAR(Y,X,G)
1080 IF G=98 OR G=101 OR G=105 THEN 1230 ELSE IF G=122 OR G=123 THEN 1460
1090 CALL GCHAR(Y+1,X,G):: IF G=113 OR G=124 THEN 890 ELSE 1480
1100 IF L=0 THEN L=1 ELSE L=0
1110 ON L+1 GOTO 1120,1130
1120 CALL HCHAR(12,22-BI,120):: CALL HCHAR(12,23-BI,121):: CALL HCHAR(13,22-BI,1
22):: CALL HCHAR(13,23-BI,123):: RETURN
1130 CALL HCHAR(12,22-BI,32,2):: CALL HCHAR(13,22-BI,32,2):: RETURN
1140 CALL HCHAR(4,16,32,2):: CALL HCHAR(5,16,32,2):: RETURN
1150 Y=Y-8 :: YB=INT((Y+15)/8):: XB=INT((X+10)/8):: CALL GCHAR(YB,XB,G):: IF G<>
124 THEN Y=Y+8 :: GOTO 890 ELSE 1170
1160 Y=Y+8 :: YY=INT((Y+15)/8):: XX=INT((X+10)/8):: CALL GCHAR(YY,XX,G):: IF G<>
124 THEN Y=Y-8 :: GOTO 890
1170 GOSUB 1050 :: CALL PATTERN(#1,60):: CALL LOCATE(#1,Y,X):: CALL POSITION(#1,
Y,X):: YB=INT((Y+15)/8):: XB=INT((X+10)/8)
1180 CALL GCHAR(YB,XB,G):: IF G<>124 THEN 890 ELSE CALL GCHAR(YB+1,XB,G):: IF G<
>124 THEN 890
1190 ! STEUERRUNG 2
1200 CALL POSITION(#1,Y,X):: CALL JOYST(1,XX,YY):: IF YY=4 THEN Y=Y-8 ELSE IF YY
=-4 THEN Y=Y+8
1210 GOTO 1170
1220 ! SETZEN DER FRESSERTEILE
1230 IF X=AA THEN 1240 ELSE IF X=BB THEN 1250 ELSE 1260
1240 XY=A(1):: YX=A(2):: AX=AA :: GOTO 1270
1250 XY=B(1):: YX=B(2):: AX=BB :: GOTO 1270
1260 XY=C(1):: YX=C(2):: AX=CC
1270 CALL SOUND(100,345,0):: IF G=98 THEN 1280 ELSE IF G=105 THEN 1380 ELSE 1420
1280 GOSUB 1100 :: IF Y-10=XY THEN 1290 ELSE IF Y-5=XY THEN 1320 ELSE 1340
1290 SC=SC+50 :: CALL HCHAR(Y,X-1,32,3):: DISPLAY AT(20,AX-3)SIZE(3):"abc"
1300 CALL GCHAR(20,7,G):: CALL GCHAR(20,17,HI):: CALL GCHAR(20,27,GV):: IF G=32
OR HI=32 OR GV=32 THEN 1430
1310 GOSUB 1450 :: CALL POSITION(#21,Y,X):: SC=SC+(X*BI*3*PH):: GOTO 220
1320 SC=SC+50 :: CALL HCHAR(Y,X-1,32,3):: DISPLAY AT(Y+5,X-3)SIZE(3):"abc"
1330 DISPLAY AT(21,AX-3)SIZE(3):"hij" :: GOTO 1430
1340 CALL GCHAR(Y+5,X,G):: CALL GCHAR(Y+10,X,GH):: IF G=32 AND GH=32 THEN 1350 E
LSE 1360
```

```
1350 SC=SC+50 :: CALL HCHAR(Y,X-1,32,3):: DISPLAY AT(Y+5,X-3)SIZE(3):"abc" :: GO
TO 1430
1360 SC=SC+100 :: CALL HCHAR(Y,X-1,32,3):: DISPLAY AT(Y+5,X-3)SIZE(3):"abc" :: D
ISPLAY AT(Y+10,X-3)SIZE(3):"hij"
1370 DISPLAY AT(22,AX-3)SIZE(3):"def" :: GOTO 1430
1380 IF Y-5=YX THEN 1390 ELSE 1400
1390 SC=SC+50 :: CALL HCHAR(Y,X-1,32,3):: DISPLAY AT(21,AX-3)SIZE(3):"hij" :: GO
TO 1430
1400 SC=SC+75 :: CALL HCHAR(Y,X-1,32,3):: CALL GCHAR(Y+5,X,G):: DISPLAY AT(Y+5,X
-3)SIZE(3):"hij" :: IF G=101 THEN 1410 ELSE 1430
1410 SC=SC+50 :: DISPLAY AT(22,AX-3)SIZE(3):"def" :: GOTO 1430
1420 SC=SC+50 :: CALL HCHAR(Y,X-1,32,3):: DISPLAY AT(22,AX-3)SIZE(3):"def"
1430 GOTO 890
1440 ! PUNKTANZEIGE
1450 SC$=STR$(SC):: FOR I=1 TO LEN(SC$):: CALL HCHAR(24,25+I,ASC(SEG$(SC$,I,1)))
:: NEXT I :: RETURN
1460 CALL SPRITE(#21,108,13,1,256,0,-1):: GOTO 890
1470 ! AUSWERTUNG,SPIELENDE?
1480 FOR I=1 TO 10 :: CALL SOUND(100,I*110,0):: NEXT I :: GOSUB 1450
1490 M=M-1 :: IF M=-1 THEN 1510 ELSE CALL HCHAR(2,2,32,7):: CALL HCHAR(2,2,95,M)
1500 CALL MOTION(#1,0,0):: CALL SPRITE(#20,128,16,1,40,#21,108,13,1,256,0,-1)::
CALL SPRITE(#1,128,16,153,6):: GOTO 890
1510 DISPLAY AT(12,7):"G A M E - O V E R"
1520 FOR I=5 TO 1 STEP -1 :: CALL SOUND(250,I*110,0,-8,6):: CALL SOUND(250,330,0
,110,7,234,10):: NEXT I
1530 CALL KEY(0,K,S):: IF S=0 THEN 1530 ELSE 200
1540 SUB MUSIK
1550 DATA 131,131,165,165,147,147,131,131,165,165,196,196,175,175,165,131,196,19
6,175,220,165,165,175,175,147,147
1560 RESTORE 1550 :: DIM Z(26):: FOR I=1 TO 26 :: READ Z(I):: NEXT I
1570 FOR I=1 TO 18 :: CALL SOUND(220,Z(I),0):: NEXT I :: CALL SOUND(1000,220,0):
: FOR I=19 TO 20 :: CALL SOUND(220,Z(I),0):: NEXT I
1580 CALL SOUND(1000,196,0):: FOR I=21 TO 26 :: CALL SOUND(220,Z(I),0):: NEXT I
:: CALL SOUND(1000,131,0)
1590 SUBEND
```

Cave-Man

Das Spiel ist auf dem TI 99/41 in TI BASIC geschrieben und wird mittels Joystick 1 gesteuert. Das Programm wurde so gestaltet, daß es auch in Extended BASIC ohne Änderungen läuft.

In Extended BASIC läuft das Programm jedoch fast doppelt so schnell als in TI BASIC. X-BASIC-Besitzer sollten das Programm auch eintippen. Es lohnt sich! Cave man zeichnet sich besonders durch die hervorragende High Resolution Grafik aus.

Schwer hatten es die Menschen in der Steinzeit, denn sie mußten sich ihr Essen bitter erkämpfen. Leo Neandertal ist nun einer dieser Menschen, der seine Familie ernähren muß.

Leo hat die Aufgabe, dem Saurier Rex die Eier zu stehlen und diese Eier in den unterirdischen Höhlen zu lagern. Freilich hat der Saurier Rex etwas gegen Leos Unterfangen und versucht mit allen Mitteln, Leo Neandertal am Diebstahl der Eier zu hindern.

Doch nun zum eigentlichen Spiel:

Tippen Sie das Programm ab, oder laden Sie dieses von Cassette oder Diskette ein.

Starten Sie das Programm jetzt mit <RUN>, und das Titelbild erscheint. Sie haben jetzt die Möglichkeit, zwischen 8 Schwierigkeitsstufen zu wählen, wobei 1 die kleinste Schwierigkeit und 8 die größte Schwierigkeit ist. Die Einstellung der Schwierigkeit erfolgt mittels Joystick und wird zu Programmbeginn vom Rechner selbst erklärt.

Nachdem das Spielbild erschienen ist, ertönt die ‚Cave-man'-Fanfare. Das ist das Zeichen, um die Schwierigkeit mittels Joystick zu wählen und danach den Aktionsknopf zu drücken, um das Spiel zu beginnen.

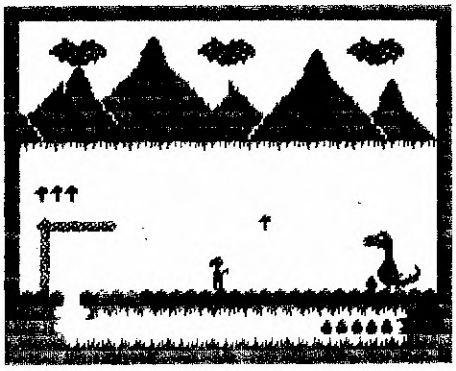
Leo Neandertal steht links im Bild unter seiner Hütte. Rechts im Bild ist der Saurier Rex zu sehen, welcher immer wieder den Kopf neigt und Feuer speiht. Steuern Sie Leo mittels des Joysticks nach rechts, bis Sie den Saurier Rex erreicht haben. Leo wird sich jetzt automatisch nach links umdrehen, und das Ei des Sauriers erscheint vor Leo. Bewegen Sie den Joystick

jetzt nach links, um Leo zusammen mit dem Ei in die Hütte zurückzuführen, damit Leo das Ei in die Höhlen werfen kann.
Sind sie jedoch im Bereich des Sauriers vorsichtig, da dieser Feuer speiht. Wenn Leo Neandertal von diesem Feuer erwischt wird, so verliert er sein Leben, und das Spiel ist zu Ende.
Leo hat jedoch eine Waffe gegen den Saurier Rex: seinen selbstgezimmerten Steinzeithammer. Leo kann diesen Hammer mitnehmen und diesen dem Saurier an den Kopf werfen, wodurch Rex für kurze Zeit betäubt wird. Dann kann Leo ungehindert das Ei stehlen.
Zu Spielbeginn stehen Leo 3 Hämmer zur Verfügung.
Wenn Leo die Absicht hat, sich mit einem Hammer auf den Weg zu Rex zu machen, so muß er in seiner Hütte stehen und der Aktionsknopf muß gedrückt werden. Leo wendet sich jetzt dem Saurier zu, und der Hammer erscheint vor Leo. Sobald Leo den Hammer jedoch genommen hat, kann er nur noch Richtung Saurier gehen, aber nicht mehr zur Hütte zurück.
Will Leo jetzt den Hammer werfen, so ist wiederum der Aktionsknopf zu drücken. Es empfiehlt sich jedoch, schon während des Fluges des Hammers weiter in Richtung Saurier zu gehen, da die Ohnmacht nur kurz anhält.

Trifft der Hammer den Saurier am Kopf, so sieht der gute Rex jetzt Sternchen. Wenn der Hammer jedoch nicht trifft oder Leo den Saurier erreicht hat, ohne den Hammer zu werfen, so wird ein Hammer abgezogen. Wenn kein Hammer mehr zur Verfügung steht, so ist das Spiel beendet.

Sobald Leo sein Leben verloren hat, ertönt ein fürchterlicher Todesschrei unseres armen Leos. Danach ertönt die ‚Cave-man'-Fanfare, und Sie können durch Druck auf den Aktionsknopf das Spiel neu beginnen.
Wenn Leo das Ei gestohlen hat, so kann er nur noch Richtung Hütte gehen und nicht mehr in die Richtung des Sauriers.
Schwierigkeitssteigerung:
Die anfänglich gewählte Schwierigkeitsstufe steigert sich nach jeweils 11 gesammelten Eiern.
Wenn es Ihnen gelingt, die gesamte Höhle mit Eiern zu füllen (22 Eier), so bekommen Sie einen Bonushammer (maximal können 6 Hämmer angezeigt werden).
Punktwertung:
Sie erhalten für jedes gesammelte Ei 10mal Schwierigkeitsstufe Punkte. Gelingt es Ihnen, die gesamte Höhle mit Eiern zu füllen, so werden Ihnen neben dem Bonushammer zusätzlich 1000mal der augenblicklichen Schwierigkeit Punkte gutgeschrieben.
Nachdem das Spiel zu Ende ist, können Sie durch Druck auf den Aktionsknopf ein neues Spiel beginnen.
Tips zum Eintippen:
Extended-BASIC-Besitzer sollten zur Eingabeerleichterung das Programm am TI BASIC-Modus eintippen, da die aufeinanderfolgenden Doppelpunkte in den Print-Anweisungen sonst als Multi-Statement-Seperatoren vom Rechner verarbeitet werden könnten.
Beim Spielablauf sollte die <ALPHA LOCK>-Taste nicht eingerastet werden, da sonst die Joystickabfrage nicht richtig funktioniert.
Wir wünschen Ihnen ein ERRORfreies Eintippen und viel Spaß beim Spiel!

```
100 REM ***************************
110 REM *    C A V E    M A N     *
120 REM *-------------------------*
130 REM * TI 99/4 A + JOYSICK 1   *
140 REM * Programmsprache :       *
150 REM * T I   B A S I C         *
160 REM *-------------------------*
170 REM * (c) 1984 by Tronicsoft  *
180 REM *              V.Becker   *
190 REM ***************************
200 REM
210 REM
220 CALL CLEAR
230 CALL SCREEN(2)
240 FOR I=1 TO 14
250 CALL COLOR(I,2,1)
260 NEXT I
270 GOSUB 3330
280 GOSUB 3470
290 RU=1
300 P=0
310 AH=3
320 SE=0
330 HF=0
340 AE=0
350 PRINT : :"  prtv  `b  prtv      prtv    9suw  ac  9suw   `b 9suw "
360 PRINT "   `b  exxd        ac          eac exxxxd  d   exxd  `b   exxxhjxxxxxd
exd exxxxd ac"
370 PRINT "hjxxxikxxxxxxxxlnxxxxxxhjxd ikxxxxxxxxxxxxxmoxxxxxxikxxd"
380 PRINT "f9f9f9ff9f99f99f9f9f9ff9f9f9"
390 FOR I=1 TO 4
400 PRINT
410 NEXT I
420 PRINT " XZZZZ[":" Y":" Y":" Y":" Y":"\]\ \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\"
430 PRINT "VVU TRSRRSSSRSSRRRRSRSRRSRVVVVU                          TVVVVVPQQPPPQPQQ
PQPQQPPPQPQQPVV":
440 DATA 2,16,2,2,13,13,11,10,5,5,15,5,9,11
```



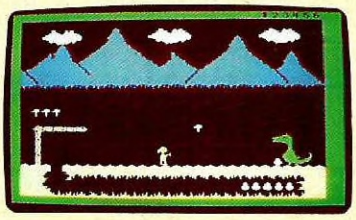
```
450 RESTORE 440
460 FOR I=1 TO 14
470 READ C
480 CALL COLOR(I,C,2)
490 NEXT I
500 CALL HCHAR(13,4,94,AH)
510 GOSUB 2470
520 GOSUB 2550
530 S=8
540 GOSUB 750
550 GOSUB 2800
560 GOSUB 3200
570 CALL VCHAR(19,26,40)
580 STV=1
590 FOR LI=1 TO 9-RU
600 CALL JOYST(1,X,Y)
610 CALL KEY(1,T,U)
620 IF (T=18)THEN 1690
630 IF HF=2 THEN 2010
640 X=X/4+2
650 ON X GOTO 820,660,980
660 IF (S>23)*(FI=1)THEN 1900
670 NEXT LI
680 IF SB>0 THEN 2310
690 STV=STV+1
700 IF STV>4 THEN 710 ELSE 720
710 STV=1
720 ON STV GOSUB 2550,2610,2670,2750
730 IF (S>23)*(FI=1)THEN 1900
740 GOTO 590
750 CALL HCHAR(24,1,78,64)
760 CALL VCHAR(1,31,78,96)
770 CALL COLOR(3,2,13)
780 CALL COLOR(4,2,13)
790 GOSUB 2990
800 GOSUB 3050
810 RETURN
820 IF HF<>0 THEN 660
830 IF (S=7)*(SE=1)THEN 1300
840 IF S=7 THEN 660
850 IF SE=1 THEN 910
860 GOSUB 2940
870 S=S-1
880 GOSUB 2870
890 IF SE=1 THEN 940
900 GOTO 660
910 CALL VCHAR(19,PE,32)
920 CALL SOUND(-10,261,0)
930 GOTO 860
940 PE=S-1
950 CALL VCHAR(19,PE,40)
960 CALL SOUND(-10,440,0)
970 GOTO 660
980 IF SE=1 THEN 660
990 IF S=25 THEN 1060
1000 IF HF=1 THEN 1150
1010 GOSUB 2940
1020 S=S+1
1030 IF HF=1 THEN 1180
1040 GOSUB 2800
1050 GOTO 660
1060 SE=1
1070 GOSUB 2940
1080 GOSUB 2870
1090 CALL VCHAR(19,26,32)
1100 CALL SOUND(-10,261,0)
1110 PE=S-1
1120 CALL HCHAR(19,PE,40)
1130 CALL SOUND(-10,440,0)
1140 GOTO 660
1150 CALL HCHAR(18,HP,32)
1160 CALL SOUND(-10,698,0)
1170 GOTO 1010
1180 HP=S+1
1190 IF HP>24 THEN 1200 ELSE 1260
1200 CALL VCHAR(13,3+AH,32)
1210 CALL SOUND(-10,110,5,112,5,114,5,-7,0)
1220 AH=AH-1
1230 IF AH=0 THEN 1900
1240 HF=0
1250 GOTO 1040
1260 HP=S+1
1270 CALL VCHAR(18,HP,94)
1280 CALL SOUND(-10,880,0)
1290 GOTO 1040
1300 FOR Z=19 TO 21
1310 CALL VCHAR(Z,6,32)
1320 CALL SOUND(-5,261,0)
1330 CALL VCHAR(Z+1,6,40)
1340 CALL SOUND(-5,440,0)
1350 CALL SOUND(-5,1046,0)
1360 NEXT Z
1370 FOR Z=6 TO 26-AE
1380 CALL VCHAR(22,Z,32)
1390 CALL SOUND(-5,261,0)
1400 CALL VCHAR(22,Z+1,40)
1410 CALL SOUND(-5,440,0)
1420 CALL SOUND(-5,523,0)
1430 NEXT Z
1440 SE=0
1450 AE=AE+1
1460 P=P+10*RU
1470 GOSUB 2990
1480 GOSUB 1520
1490 CALL VCHAR(19,26,40)
1500 CALL SOUND(-10,440,0)
1510 GOTO 660
1520 IF AE=11 THEN 1640
1530 IF AE=22 THEN 1550
1540 RETURN
1550 AH=AH+1
1560 IF AH>6 THEN 1570 ELSE 1580
1570 AH=6
1580 CALL HCHAR(13,4,94,AH)
1590 GOSUB 3200
1600 P=P+1000*RU
1610 GOSUB 2990
1620 CALL HCHAR(22,6,32,22)
1630 AE=0
1640 RU=RU+1
1650 IF RU>8 THEN 1660 ELSE 1670
1660 RU=8
1670 GOSUB 3290
1680 RETURN
1690 IF (SE=1)+(HF=2)THEN 640
1700 IF HF=1 THEN 1800
1710 IF (HF=0)*(S<9)THEN 1730
1720 GOTO 640
1730 GOSUB 2940
1740 GOSUB 2800
1750 HF=1
1760 HP=S+1
1770 CALL VCHAR(18,HP,94)
1780 CALL SOUND(-10,880,0)
1790 GOTO 640
1800 IF S<8 THEN 640
1810 FW=(12/(9-RU))/8
1820 HF=2
1830 CALL VCHAR(18,HP,32)
1840 CALL SOUND(-10,698,0)
```

```
1850 SH=1
1860 FH=18-SH
1870 CALL VCHAR(FH,HP,94)
1880 CALL SOUND(-10,880,0)
1890 GOTO 640
1900 FOR I=1100 TO 500 STEP -10
1910 CALL SOUND(-100,I,0,I-2,0,I+2,0)
1920 NEXT I
1930 GOSUB 3200
1940 CALL KEY(1,T,U)
1950 IF U=0 THEN 1940
1960 CALL CLEAR
1970 FOR I=1 TO 14
1980 CALL COLOR(I,2,1)
1990 NEXT I
2000 GOTO 290
2010 CALL VCHAR(FH,HP,32)
2020 CALL SOUND(-10,698,0)
2030 IF FH=11 THEN 2040 ELSE 2050
2040 SH=-SH
2050 FH=FH-SH
2060 HP=HP+FW
2070 CALL VCHAR(FH,HP,94)
2080 CALL SOUND(-10,880,0)
2090 IF (HP>26)*(FH>15)THEN 2220
2100 IF HP>27 THEN 2130
2110 IF FH>18 THEN 2130
2120 GOTO 640
2130 CALL VCHAR(FH,HP,32)
2140 CALL SOUND(-10,698,0)
2150 CALL VCHAR(13,3+AH,32)
2160 CALL SOUND(-10,110,5,112,5,114,5,-7,0)
2170 CALL VCHAR(19,26,40)
2180 AH=AH-1
2190 IF AH=0 THEN 1900
2200 HF=0
2210 GOTO 640
2220 HF=0
2230 FI=0
2240 SB=6
2250 CALL VCHAR(FH,HP,32)
2260 CALL SOUND(-10,698,0)
2270 GOSUB 2750
2280 GOSUB 2610
2290 STV=2
2300 GOTO 640
2310 CALL HCHAR(15,26,32,2)
2320 STS=STS+1
2330 IF STS>2 THEN 2340 ELSE 2350
2340 STS=1
2350 ON STS GOTO 2400,2430
2360 SB=SB-1
2370 IF SB=0 THEN 2380 ELSE 740
2380 CALL HCHAR(15,26,32,2)
2390 GOTO 740
2400 CALL HCHAR(15,26,140,2)
2410 CALL SOUND(-10,1396,0)
2420 GOTO 2360
2430 CALL VCHAR(15,26,141)
2440 CALL VCHAR(15,27,143)
2450 CALL SOUND(-10,1046,0)
2460 GOTO 2360
2470 REM Saurier Korpus
2480 CALL VCHAR(18,27,72)
2490 CALL VCHAR(19,27,73)
2500 CALL VCHAR(18,28,74)
2510 CALL VCHAR(19,28,75)
2520 CALL VCHAR(18,29,76)
2530 CALL VCHAR(19,29,77)
2540 RETURN
2550 REM Saurier Kopf gerade
2560 CALL VCHAR(16,26,68)
2570 CALL VCHAR(17,26,69)
2580 CALL VCHAR(16,27,70)
2590 CALL VCHAR(17,27,71)
2600 RETURN
2610 REM Saurier Kopf geneigt
2620 CALL VCHAR(16,26,64)
2630 CALL VCHAR(17,26,65)
2640 CALL VCHAR(16,27,66)
2650 CALL VCHAR(17,27,67)
2660 RETURN
2670 REM Feuer des Saurier
2680 FI=1
2690 CALL VCHAR(18,24,132)
2700 CALL VCHAR(19,24,133)
2710 CALL VCHAR(18,25,134)
2720 CALL VCHAR(19,25,135)
2730 CALL SOUND(-10,245,0,246,0,247,0,-5,0)
2740 RETURN
2750 REM Feuer loeschen
2760 FI=0
2770 CALL VCHAR(18,24,32,2)
2780 CALL VCHAR(18,25,32,2)
2790 RETURN
2800 REM Cave Man nach rechts
2810 CALL VCHAR(18,S,128)
2820 CALL VCHAR(19,S,129)
2830 CALL VCHAR(17,S,137)
2840 CALL VCHAR(18,S-1,138)
2850 CALL SOUND(-10,523,0)
2860 RETURN
2870 REM Cave Man nach links
2880 CALL VCHAR(18,S,130)
2890 CALL VCHAR(19,S,131)
2900 CALL VCHAR(17,S,139)
2910 CALL VCHAR(18,S+1,136)
2920 CALL SOUND(-10,523,0)
2930 RETURN
2940 REM Cave Man loeschen
2950 CALL VCHAR(17,S,32,3)
2960 CALL HCHAR(18,S-1,32,3)
2970 CALL SOUND(-10,349,0)
2980 RETURN
2990 P$=STR$(P)
3000 FOR LG=1 TO LEN(P$)
3010 PR=ASC(SEG$(P$,LG,1))
3020 CALL HCHAR(1,30-LEN(P$)+LG,PR)
3030 NEXT LG
3040 RETURN
3050 CALL VCHAR(18,7,ASC(STR$(RU)))
3060 GOSUB 3200
3070 CALL JOYST(1,X,Y)
3080 IF (X=0)*(Y=0)THEN 3130
3090 RU=RU+1
3100 IF RU>8 THEN 3110 ELSE 3120
3110 RU=1
3120 CALL VCHAR(18,7,ASC(STR$(RU)))
3130 FOR VERZ=1 TO 10
3140 CALL KEY(1,T,U)
3150 IF T=18 THEN 3180
3160 NEXT VERZ
3170 GOTO 3070
3180 CALL VCHAR(18,7,32)
3190 RETURN
3200 CALL SOUND(200,523,0,659,0)
3210 CALL SOUND(100,391,0)
3220 CALL SOUND(100,440,0)
3230 CALL SOUND(200,523,0)
3240 CALL SOUND(200,587,0)
```

```
3250 CALL SOUND(200,523,0)
3260 CALL SOUND(200,329,0,391,0)
3270 CALL SOUND(200,391,0,523,0)
3280 RETURN
3290 CALL SOUND(200,391,0,523,0,659,0)
3300 CALL SOUND(200,440,0,523,0,698,0)
3310 CALL SOUND(200,391,0,523,0,659,0)
3320 RETURN
3330 PRINT "      C A V E    M A N              ----------------": : :
3340 PRINT "Schwierigkeitswahl :": :"Druecke roten Aktionsknopf. Warte bis Fanfa
re ertoent."
3350 PRINT "Bewege Joystick 1 so lange, bis die gewuenschte"
3360 PRINT "Schwierigkeit im gruenen    Feld unter der Huette        erscheint."
3370 PRINT "Lasse dann Joystick los und druecke erneut Aktionsknopf.": : :
3380 PRINT "1 - kleinste Schwierigkeit  8 - groesste Schwierigkeit": : :"    @ 1
984 by Tronicsoft"
3390 CALL CHAR(64,"3C4299A1A199423C")
3400 CALL SCREEN(4)
3410 GOSUB 3200
3420 CALL KEY(1,T,U)
3430 IF U=0 THEN 3420
3440 CALL SCREEN(2)
3450 CALL CLEAR
3460 RETURN
3470 REM Grafik umbelegen
3480 DATA ,0010101010010001,00282828,0028287C287C2828,0038545038145438,0060640810
204C0C,0020505020544834
3490 DATA 0008081,000C1E1E3F3F3F1E,0020100808008102,000028107C1028,000010107C101,
0000000000030102,0000000007C
3500 DATA 0000000000000303,0000040810204,00182424242418,0008180808080808,0018240408
103C,00182408042418,00282828380808
3510 DATA 003C2038042418,00182038242418,003C040810101,00182418242418,001824241C0
418,0000303000303,0000303000030102
3520 DATA 0008102040201008,0000007C007C,0020100804008102,0038440408100001,00000001
03060C0F,1F1F3F3E7C78E04,3878FCFEFF7FFFFF
3530 DATA FFFF9F0F0F0F0F0F,0000001F7FFF3FFF,7F,0078FCFE9FFFFFFF,FFFF1F0F0F0F0F0F
,0F1F1F3F7FFFFFFF,FF7F7F3F1F0F060E
3540 DATA 008080C0C0C0E0E8,FCF8F1FFFFFF367,0000000080C06060E,5E7CFCF8F08,FFFFFFFF
FFFFFFFF,,00000282A3E7FFFF
3550 DATA 000001092B6FFFFF,FFFFE7C5414,FFFFF6D4908,0F3F0703030F071F,F8E0F0C0C0E0
FCF,FFFFFFFFFFFFFFFF,004444445454542B
3560 DATA 00040F1B3E6F3D17,171F1B1F161F0B1D,000000BFEEBB7FEF,000000C060F0A0C,187
EFFFFFFFFFFFFF,1F7FFFFFFFFFFFFF,183E7EFE18181818
3570 DATA 0000000000000007C,010307070F0F1F1F,3F7F7F3F7F7FFFFF,80C0E0E0E0F0F8F8,F8
FCFCFEFCFEFFFF,80E0E0F0F4FCFEFF,0107070F2F3F7FFF
3580 DATA FFB9B1111,D3921202,0387EFF7FBFBF9FE,FFFFFFFFFFFFFFFF,FFFFFFFFFFFFFFFF,
7F3FDFDFCFF7F7FF,FFFFFFFFFFFFFFFF
3590 DATA FEFCFBFBF3EFEFFF,C0E1F7EFDFDF9F7F,FFFFFFFFFFFFFFFF,0001010107 1F3DFF,FF
FF1F070703,00C0E0F8FCDFFFFF,DDFFFFEFFF3F1F0C
3600 DATA 0000337F7FFFFFFF,FBFFFFEFFFFCF88,00008080E0F8BCFE,F6FE7EECE0C08,FFFFFF
FFFFFFFFFF,0000004428101 01,0000007C0810207C
3610 DATA 0018202040202018,0010101000101 01,0030008080408083,0000205408,,E8FCF8706
177FDF8,F8F8F8F8D8D8D8FC
3620 DATA 173F1F0E86EEBF1F,1F1F1F1F1B1B1B3F,,0103070F1F3F7FFF,0103060C1C3878F,F0
E0E0C0C0808,80C0C0C
3630 DATA 0000000000000070F8,01030303,0000000000000E1F,0000004428102844,0000080402
030408,,0000204080804 02
3640 RESTORE 3480
3650 FOR I=32 TO 143
3660 READ C$
3670 CALL CHAR(I,C$)
3680 NEXT I
3690 RETURN
```

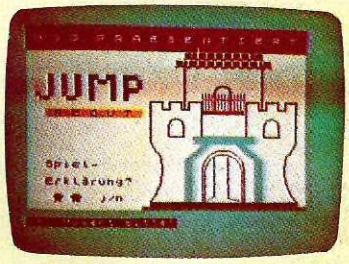
Kassettenservice

Cave Man:



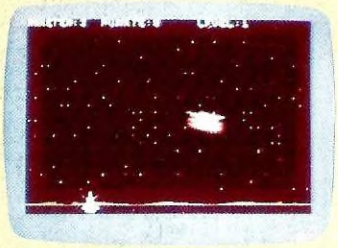
Leo Neandertal lebt in der Steinzeit. Schwer hatten es die Menschen in der Steinzeit, denn sie mußten sich ihr Essen bitter erkämpfen. Leo will dem Saurier Rex die Eier stehlen und diese in den unterirdischen Höhlen lagern. Der Saurier Rex aber hat etwas gegen Leos Unterfangen und versucht mit allen Mitteln, Leo Neandertal am Diebstahl zu hindern. Ein Spiel für den **TI 99.**

Jump about:



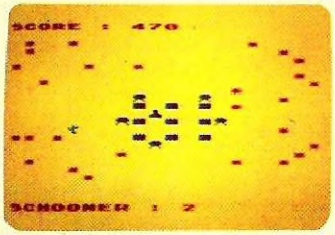
Ein kleiner Floh und sein Freund haben ein schweres Schicksal zu meistern. Sie sind beim Herumhüpfen unbewußt in den Einflußbereich eines bösen Magiers geraten. Nun gilt es, den Weg in die Freiheit zu finden. Ein Spiel für den **ZX-Spectrum** mit toller Graphik.

Monster-Attak:



Für den **Commodore 64.** Fremde Wesen greifen die Erde an. Versuchen Sie die angreifenden Monster vor Erreichen der Erde zu zerstören. Das Spiel unterteilt sich in 6 Level, wobei bei jedem Level die Spielgeschwindigkeit sowie die erreichbare Punktzahl ansteigt. Das Spiel wird mit Joystick gespielt.

Tomstone-City:



Eine kleine Stadt im „Wilden Westen" gewährt Dir Schutz vor Deinen Feinden. Solange Du sie nicht verläßt kann Dir nichts passieren. Außerhalb der Stadt wirst Du jedoch gejagt. Gewählt werden kann zwischen verschiedenen Schwierigkeitsgraden. Ein Spiel für **Atari 600/800 XL.**

Maya:

Das Spiel läuft auf dem **TI-99/4 A** mit dem **Extented Basic Modul** und wird mittels Joystick gesteuert. Der Rechner kann in diesem Spiel bis zu 378!! verschiedene Bildschirmszenen darstellen. Das Spiel verfügt über eine hervorragende grafische Darstellung. Begleiten Sie also Coconut Joe auf die abenteuerliche Expedition in den Urwald. Vielleicht haben Sie Glück und finden die Schätze der Mayas.

In eigener Sache:

Wir haben uns verändert!

Unter dieser Rubrik, liebe Leser, wollen wir Ihnen unser neues Konzept für das kommende Jahr 1985 vorstellen. Nachdem wir nun schon seit April dieses Jahres auf dem Zeitschriftenmarkt erfolgreich mitmischen, können wir eine erste Bilanz ziehen. Das Resultat wird – wie wir meinen – viele Leser mit Sicherheit erfreuen.

Fest steht: Wir bleiben unseren treuen Lesern und Hobby-Programmierern erhalten, aber: in neuer Form!

Unser Schwerpunkt wird weiterhin bei Software-Listings der Spitzenklasse liegen. Berücksichtigt werden dabei: *Commodore 64, VC-20, TI-99, Atari, Apple II, ZX-Spectrum* und der *ZX-81*. Zusätzlich erhält der Commodore 64 in jeder unserer Ausgabe einen umfangreichen *Sonderteil!*

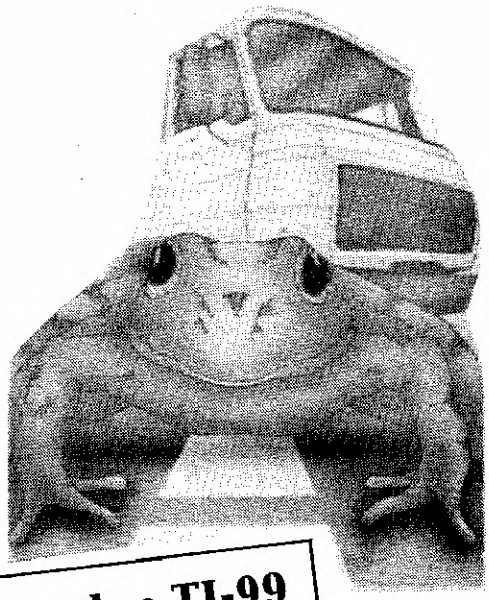
Ferner ist Ihnen sicherlich nicht entgangen, daß die jetzige Ausgabe von „Computronic" eine Doppelausgabe ist. Aus Gründen der *zu kurzen Angebotszeit* unserer Zeitschrift im Handel haben wir uns entschlossen, alle 2 Monate auf dem Markt zu erscheinen. Ihr Plus dabei: Wir bringen acht Seiten mehr, Aktuelles, viele Listings, komplette Kurse und andere tolle Überraschungen zum Preis von nur DM 6,50.

Wir jedenfalls hoffen, mit unserem Konzept dem Markt gerecht zu werden, und wünschen uns, liebe Leser, für jeden immer etwas Interessantes dabeizuhaben.

Ihre Redaktion.

Topprogramm

Frogger



jetzt auch für den TI-99 mit Ext. Basic-Modul

Während des Spielverlaufes müssen Sie versuchen, alle 10 Frösche, die Ihnen zur Verfügung stehen, in ihren Unterschlupf zu bringen.

Zunächst muß jeder Frosch eine vielbefahrene Straße überqueren. Auf dem Mittelweg angekommen, kann er sich erst einmal ausruhen und die Flußüberquerung vorbereiten. – Ab Level 6 wird die Ruhepause auf dem Mittelweg durch eine sich hin- und herbewegende Schlange erschwert. – Auf die andere Seite des Flusses kann der Frosch nur gelangen, wenn er abwärtstreibende Schildkröten und Baumstämme zu Hilfe nimmt. Auf ihnen kann er sich, ohne ins Wasser zu müssen, bewegen.
Ist der Fluß erfolgreich genommen worden, steht das nächste Hindernis bevor. Eine Start- und Landebahn für Flugzeuge muß gekreuzt werden. Aber auch das dürfte für die „echten Könner" kein Problem sein. Sind alle 10 Frösche gut untergebracht, erfahren Sie über den Score, wieviel Zeit dafür benötigt wurde. Das Spiel ist in 99 Levels unterteilt. 1–5 ist für Anfänger, 6–20 für Amateure und 21–99 für Profis. Gesteuert wird mit Joystick.

```
100 ! ######################
110 ! #                    #
120 ! # SPECIAL - FROGGER #
130 ! #                    #
140 ! #A GAME FOR 1 PLAYER#
150 ! #                    #
160 ! #   (C) BY KABOSO    #
170 ! #  KARSTEN BOGUSCH   #
180 ! #     SOFTWARE       #
190 ! #                    #
200 ! #                    #
210 ! # D-4600 DORTMUND 41#
220 ! #                    #
230 ! #                    #
240 ! ######################
250 !
260 !
270 !
280 ON BREAK NEXT :: ON WARNING NEXT :: CALL SCREEN(2):: CALL CLEAR :: CALL MAGN
IFY(3):: RANDOMIZE :: DIM J(14)
290 FOR I=0 TO 8 :: CALL COLOR(I,4,2):: NEXT I :: CALL COLOR(2,15,1,3,16,2,4,16,
2,9,4,13,10,6,6,11,8,8,14,16,1)
300 DISPLAY AT(12,1):"ICH DEFINIERE MEINE ZEICHEN!"
310 FOR I=65 TO 90 :: CALL SOUND(-4200,110+I*2,0,112+I*2,0,114+I*2,0):: READ A$
:: CALL CHAR(I,A$):: NEXT I :: GOSUB 540
320 DATA 003F666C746466EF,007E313B3633337E,003E634140603F1E,007E321119113F7E,007
F33303C38317F,007F723239383078
330 DATA 003E6648504F663E,007C603F3132327A03,007C38181C1A183C2,003F13060606263F,
007B72363C363673,007830303072637F
340 DATA 007F2B2A49494963,007C76323A323477,003E634559617F3E,007E33313F3E3078,003
E634559617F3E07,007F313A34323179,003E413C06634F3E
350 DATA 007F492A0808081C,006F2626666E7E37,007362232321E0C,00772222414936,00793
```

```
2140814264F,007341623418183C,007F43260C1A317F
360 CALL CLEAR :: CALL CHAR(40,"FFFFFFFFFFFFFFFF",90,"FFFFFFFFFFFFFFFF",89,"00FF
FFFFFFFFFF00",100,"B588254AA10C5184",140,"4224183C3C3C4218")
370 CALL CHAR(141,"18187E7E18181818")
380 CALL CHAR(48,"00D68282828282D6",49,"0010101010101010",50,"00D60202D68080D6",
51,"00D60202160202D6",52,"00828282D6020202")
390 CALL CHAR(53,"00D68080D60202D6",54,"00808080D68282D6",55,"00D6020202020202",
56,"00D68282D68282D6",57,"00D68282D6020202")
400 CALL CLEAR :: CALL DELSPRITE(ALL):: A=0 :: FOR I=1 TO 10 :: A=A+16 :: CALL S
PRITE(#I,136,INT(RND*14+3),A,247,-14,0):: NEXT I
410 DISPLAY AT(1,3):"ᲂᲂᲂ ᲂᲂᲂ ᲂᲂᲂ ᲂᲂᲂ ᲂᲂᲂ ᲂᲂᲂ ᲂᲂᲂ Z   Z Z Z Z Z   Z   Z   Z Z ᲂᲂ
 ᲂᲂᲂ ᲂ ᲂ ᲂ ᲂ ᲂ ᲂ ᲂᲂ  ᲂᲂᲂ Z   ZZ  Z Z Z Z Z Z Z   ZZ  ᲂ   ᲂ ᲂ ᲂᲂᲂ ᲂᲂᲂ ᲂᲂᲂ ᲂᲂᲂ ᲂ ᲂ
"
420 DISPLAY AT(9,3):"ᲂᲂᲂ ᲂᲂᲂ ᲂᲂᲂ ᲂᲂᲂ  ᲂ  ᲂᲂᲂ ᲂ   Z   Z Z Z   Z    Z  Z Z Z   ᲂᲂᲂ
 ᲂᲂᲂ ᲂᲂ  ᲂ    ᲂ  ᲂᲂᲂ ᲂ     Z Z   Z   Z    Z  Z Z Z   ᲂᲂᲂ ᲂ   ᲂᲂᲂ ᲂᲂᲂ  ᲂ  ᲂ ᲂ ᲂᲂᲂ
"
430 DISPLAY AT(17,1):"MIT WELCHEN SCHWIERIGKEITS-":"":"GRAD WOLLEN SIE BEGINNEN
?":"":"01-05 --ANFAENGER--":"06-20 ---AMATEUR---":"21-99 -----PROFI----"
440 DISPLAY AT(24,1):"ENTSCHEIDEN SIE SICH !" :: ACCEPT AT(24,24)BEEP VALIDATE(D
IGIT)SIZE(2):V :: CALL CLEAR :: CALL DELSPRITE(ALL)
450 V=V-1 :: M=10 :: Q,R,PU,W=0
460 DISPLAY AT(2,2):"SCORE:" :: DISPLAY AT(2,14):"HIGHSCORE:" :: DISPLAY AT(2,24
):STR$(HI)
470 CALL HCHAR(1,1,100,32):: CALL HCHAR(22,1,100,32):: CALL HCHAR(2,1,100,3):: C
ALL HCHAR(2,31,100,2):: CALL HCHAR(23,1,100,3)
480 CALL HCHAR(4,1,40,576):: CALL HCHAR(23,10,89,20):: CALL HCHAR(24,19,100,4)::
 CALL HCHAR(7,1,100,32):: CALL HCHAR(12,1,100,32)
490 CALL HCHAR(24,1,100,3):: CALL HCHAR(24,31,100,2):: CALL HCHAR(24,19,100,4)::
 CALL HCHAR(2,15,100)
500 CALL HCHAR(21,1,100,32):: CALL HCHAR(3,1,100,32):: CALL HCHAR(4,1,100,3):: C
ALL HCHAR(4,5,100,5):: CALL HCHAR(4,11,100,5)
510 CALL HCHAR(4,17,100,5):: CALL HCHAR(4,23,100,5):: CALL HCHAR(4,29,100,4):: C
ALL HCHAR(23,31,100,2):: CALL HCHAR(8,1,112,128)
520 DISPLAY AT(24,2)SIZE(-6):"FROGS:" :: CALL HCHAR(24,10,136,9):: DISPLAY AT(24
,21)SIZE(-5):"GAME:" :: DISPLAY AT(24,27)SIZE(-2):STR$(V+1):: DISPLAY AT(23,2):"
TIME :"
530 CALL HCHAR(23,10,89,20):: CALL HCHAR(24,19,100,4):: GOTO 630
540 CALL CHAR(36,"00000001026 2E3180000000000000000001894D5730100000000000000000
00")
550 CALL CHAR(96,"001C1C3F62EEDEDADEEE623F1C1C0000003838FEFEABFFABFFABFEFE383800
00")
560 CALL CHAR(104,"00FF80FF80FF80FF80FF80FF80FF80FF00FF01FF01FF01FF01FF01FF01FF0
1FF")
570 CALL CHAR(112,"003F7FFFFF7F3F0F0F3F7FFFFF7F3F0000FCFEFFFFFEFCF0F0FCFEFFFFFEF
C00")
580 CALL CHAR(120,"00001C7FD3717571717571D37F1C000000001CFEE7E6E6E6E6E6E6E7FE1C0
000")
590 CALL CHAR(124,"5555552A7F7FE0BEBEE07F7F2A5555550704070CF714CFC4C4CF14F70C070
407")
600 CALL CHAR(128,"009CDCDCC8DFDCD9FBF9DCDFC8DCDC9C0038383813FB7F3F1F3F7FFB13383
838")
610 CALL CHAR(132,"0000000001033FFFFF3F0301000000000003070F0F1F3FFFFFFFFF3F1F0703
000")
620 CALL CHAR(136,"18423C3C3C18244200000000000000000000000000000000000000000000
000"):: RETURN
630 CALL SPRITE(#1,96,5,145,256,#2,96,5,145,170,#3,96,5,145,85)
640 CALL SPRITE(#4,124,7,129,220,#5,124,7,129,135,#6,124,7,129,40)
650 CALL SPRITE(#24,120,14,113,175,#25,120,14,113,100,#26,120,14,113,10)
660 CALL SPRITE(#10,128,9,97,160,#11,128,9,97,70,#12,128,9,97,250)
670 CALL SPRITE(#20,112,2,73,250,#21,112,2,73,100)
680 CALL SPRITE(#19,104,13,57,200,#18,104,13,57,120,#17,104,13,57,50)
690 CALL SPRITE(#15,132,8,33,250,#22,132,8,33,125)
700 FOR I=4 TO 28 STEP 6 :: CALL HCHAR(4,I,32):: NEXT I :: GOTO 1230
```

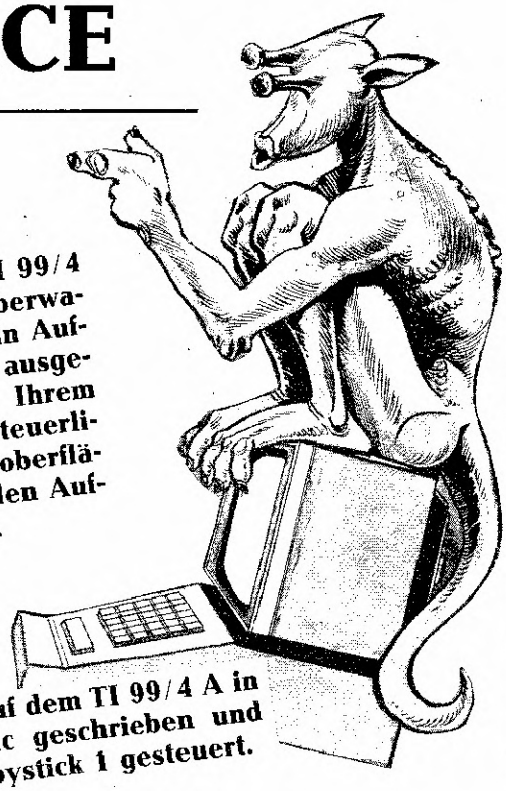
```
710 R=0 :: CALL SOUND(100,210,1,230,1,330,1,-2,1):: IF M=0 THEN 1330 ELSE IF M<1
0 THEN CALL HCHAR(24,9+M,141)
720 ON ERROR 710
730 L=161 :: O=INT(RND*160)+40
740 CALL SPRITE(#16,136,16,L,O)
750 CALL JOYST(2,Y,X):: T=T+1 :: CALL HCHAR(23,30-INT(T/5),32):: IF T=100 THEN 1
090
760 IF X=4 THEN L=L-8 ELSE IF Y=4 THEN O=O+8 ELSE IF Y=-4 THEN O=O-8 ELSE IF X=-
4 AND L>160 THEN 750 ELSE 780
770 GOTO 820
780 IF X=-4 THEN L=L+8 :: GOTO 820
790 GOTO 800
800 CALL COINC(ALL,B):: FC=FC+1 :: IF B=-1 THEN 1090 ELSE IF FC=16 THEN 810 ELSE
 750
810 FC=0 :: H=-1*H :: GF=-H :: CALL MOTION(#28,0,H*(INT(RND*15)+5)):: CALL MOTIO
N(#27,0,GF*(INT(RND*27)+6)):: GOTO 750
820 IF L<89 THEN 890 ELSE CALL LOCATE(#16,L,O):: W=W+1 :: CALL SOUND(-5,110,8,10
00,5,-2,5):: R=R+SGN(X):: CALL COINC(ALL,B):: IF B=-1 THEN 1090 ELSE 750
830 IF L=89 THEN CALL MOTION(#16,0,0):: GOTO 750
840 CALL JOYST(2,X,Y):: T=T+1 :: CALL HCHAR(23,30-INT(T/5),32):: IF T=100 THEN 1
090
850 IF Y=4 THEN 890 ELSE IF Y=-4 THEN 860 ELSE IF X=-4 THEN 920 ELSE IF X=4 THEN
 940 ELSE 830
860 CALL POSITION(#16,L,O):: R=R-1
870 L=L+8 :: CALL LOCATE(#16,L,O):: W=W+5 :: CALL MOTION(#16,0,J(R)):: CALL SOUN
D(-5,110,8,1000,5,-2,5):: CALL COINC(ALL,B):: IF B=-1 THEN 830 ELSE 880
880 IF L=89 THEN 830 ELSE 1090
890 CALL POSITION(#16,L,O):: R=R+1
900 L=L-8 :: CALL LOCATE(#16,L,O):: W=W+5 :: CALL MOTION(#16,0,J(R)):: CALL SOUN
D(-5,110,8,1000,5,-2,5):: CALL COINC(ALL,B):: IF B=-1 THEN 830 ELSE 910
910 IF L=49 THEN 950 ELSE 1090
920 CALL POSITION(#16,L,O):: O=O-8
930 CALL LOCATE(#16,L,O):: W=W+1 :: CALL SOUND(-5,110,8,1000,5,-2,5):: CALL COIN
C(ALL,B):: IF B=0 THEN 1090 ELSE 830
940 CALL POSITION(#16,L,O):: O=O+8 :: W=W+5 :: CALL LOCATE(#16,L,O):: CALL SOUND
(-5,110,8,1000,5,-2,5):: CALL COINC(ALL,B):: IF B=0 THEN 1090 ELSE 830
950 CALL MOTION(#16,0,0):: CALL LOCATE(#16,INT(L/8)*8+1,INT(O/8)*8+8):: W=W+5 ::
 L=INT(L/8)*8+1 :: O=INT(O/8)*8+8
960 !
970 CALL JOYST(2,X,Y):: T=T+1 :: CALL HCHAR(23,30-INT(T/5),32):: IF T=100 THEN 1
090
980 IF L=33 AND Y=4 THEN GOSUB 1030 ELSE IF Y=-4 AND L=49 THEN 840
990 IF Y=4 THEN L=L-8 ELSE IF X=4 THEN O=O+8 ELSE IF X=-4 THEN O=O-8 ELSE IF Y=-
4 THEN L=L+8 ELSE 1010
1000 GOTO 1070
1010 CALL COINC(ALL,B):: IF B=-1 THEN 1090
1020 GOTO 970
1030 CALL GCHAR(4,INT(O/8)+1,D):: IF D<>32 THEN 1090
1040 PU=PU+200 :: PP=PU+W :: DISPLAY AT(2,8)SIZE(-5):STR$(PP):: IF PU+W>HI THEN
DISPLAY AT(2,24)SIZE(-5):STR$(PP)
1050 CALL POSITION(#16,L,O):: CALL DELSPRITE(#16):: CALL HCHAR(L/8,(O/8)+1,140):
: GOTO 1140
1060 RETURN
1070 CALL LOCATE(#16,L,O):: W=W+1 :: CALL SOUND(-5,110,8,1000,5,-2,5):: CALL COI
NC(ALL,B):: IF B=-1 THEN 1090 ELSE 960
1080 !
1090 CALL DELSPRITE(#16):: GOSUB 1360 :: M=M-1 :: PP=PU+W :: DISPLAY AT(2,8)SIZE
(-5):STR$(PP):: T=0
1100 IF PU+W>HI THEN DISPLAY AT(2,24)SIZE(-5):STR$(PP)
1110 IF M=0 THEN 1330
1120 CALL HCHAR(23,10,89,20):: FOR X=1 TO 20 :: NEXT X :: GOTO 710
1130 !
1140 FOR Z=1 TO 10 :: CALL SOUND(-90,4000,4,3900,6):: CALL SOUND(-40,3000,2,2900
```

```
,4):: NEXT Z
1150 Q=Q+1 :: IF Q=5 THEN 1170 :: T=0
1160 CALL HCHAR(23,10,89,20):: GOTO 710
1170 DISPLAY AT(24,2)SIZE(-5):"FROGS:" :: CALL HCHAR(24,10,136,9)
1180 !FOR X=1 TO 4 :: CALL SOUND(120,450,2,460,3):: CALL SOUND(120,780,3,790,3):
: CALL SOUND(120,670,3,680,2):: NEXT X
1190 FOR S=500 TO 110 STEP -50 :: CALL SOUND(1,5*S,0):: NEXT S :: FOR S=110 TO 5
00 STEP 50 :: CALL SOUND(1,3*S,0):: NEXT S
1200 M=10 :: Q,T=0 :: CALL HCHAR(23,10,89,20)
1210 V=V+1 :: IF V>40 THEN V=40
1220 DISPLAY AT(24,27)SIZE(-2):STR$(V+1):: CALL SOUND(390,780,5,790,6,770,7,-1,0
)
1230 FOR I=4 TO 28 STEP 6 :: CALL HCHAR(4,I,32):: NEXT I :: FOR X=1 TO 3 :: CALL
 MOTION(#X,0,-(V+7)):: NEXT X :: FOR X=4 TO 8 :: CALL MOTION(#X,0,V+5):: NEXT X
1240 FOR X=24 TO 26 :: CALL MOTION(#X,0,-(V+8)):: NEXT X
1250 FOR X=10 TO 12 :: CALL MOTION(#X,0,V+9):: NEXT X
1260 CALL MOTION(#15,0,-(V+18),#22,0,-(V+18))
1270 J(10)=-(V+5):: J(11)=-(V+5):: J(12)=V+7 :: J(13)=V+7
1280 FOR X=20 TO 21 :: CALL MOTION(#X,0,J(10)):: NEXT X :: FOR X=17 TO 19 :: CAL
L MOTION(#X,0,J(12)):: NEXT X
1290 IF V<5 THEN 1320 ELSE H=INT(RND*2):: IF H=0 THEN H=-1
1300 CALL SPRITE(#28,36,15,89,200,0,H*(INT(RND*22)+5))
1310 IF V<7 THEN 1320 ELSE GF=-H :: CALL SPRITE(#27,36,15,49,50,0,GF*(INT(RND*28
)+5))
1320 GOTO 710
1330 HI=MAX((PU+W),HI):: DISPLAY AT(23,2)BEEP:"ddd NOCH EIN SPIEL? J/N ddd":"ddd
dddddddddddddddddddddddddddd"
1340 CALL KEY(0,K,S):: IF S=0 THEN 1340
1350 IF K=74 OR K=106 THEN 400 ELSE IF K=78 OR K=110 THEN CALL CLEAR :: BREAK EL
SE 1340
1360 CALL SOUND(250,184,8,369,3,739,10):: CALL SOUND(125,195,8,391,3,783,10)
1370 CALL SOUND(125,184,8,369,3,738,10):: CALL SOUND(250,146,8,293,3,587,10)
1380 CALL SOUND(250,138,8,277,3,554,10):: CALL SOUND(1000,123,8,246,3,493,10)::
RETURN
```

MOON RACE

Abenteuer auf dem Mond

Alarm auf der Mondbasis TI 99/4 A. In der von Ihnen zu überwachenden Sektion Beta ist ein Aufstand der Mondlebewesen ausgebrochen. Starten Sie mit Ihrem Moon Racer zu einer abenteuerlichen Fahrt über die Mondoberfläche und zerschlagen Sie den Aufstand der Mondbewohner.



Das Spiel ist auf dem TI 99/4 A in Extended Basic geschrieben und wird mittels Joystick 1 gesteuert.

Die einzigartige Möglichkeit des TI 99/4 A, bis zu 28 Sprites vom Basic ansprechen zu können, erlaubt es, bei diesem Spiel drei verschiedene Grafikebenen in verschiedenen Geschwindigkeiten derart zu verschieben, daß das Bild plastisch wirkt. Dieser Effekt ist besonders beim Einsatz eines Farbfernsehers zu beobachten. Durch diese Maßnahme mußte jedoch der Vergrößerungsfaktor 4 für die Sprites gewählt werden, wodurch die Grafik eine etwas größere Auflösung zeigt. Dies tut jedoch dem Reiz des Spieles keinen Abbruch.

Tippen Sie das Programm ein und starten Sie dieses mit RUN. Nach kurzer Initialisierungsphase erscheint das Titelbild auf dem Bildschirm. Die Titelmelodie erklingt und die beiden hinteren Grafikebenen setzen sich in Bewegung. Diese Bewegung wird während des ganzen Spieles unaufhörlich fortgeführt.
Sie haben jetzt die Aufgabe, mit Ihren Moon Racer die Sektion Beta zu patrouillieren. Dazu steht Ihnen eine Anzeige von mehreren Informationen zur Verfügung: Am oberen Bildschirmrand ist links hinter den Schriftzug >Zone< die Patrouillenzone eingeblendet, in welcher Sie sich befinden. Die Sektion Beta besteht aus 4 Zonen.
In der Mitte sehen Sie die Ihnen zur Verfügung stehenden Racer (anfänglich 5). Daneben wird hinter dem Schriftzug >Score< Ihr aktueller Punktestand angezeigt. Am unteren linken Bildschirmrand ist der Level eingeblendet, in welchem Sie sich befinden. Da Sie nach jedem Unglück, das Ihnen widerfährt, bei der Mondbasis neu mit der Patrouille beginnen müssen, erhöht sich der Level immer dann, wenn es Ihnen gelungen sein sollte, mit einem Fahrzeug sämtliche 4 Zonen der Sektion Beta zu durchfahren bzw. zu durchfliegen.

Auf dem Titelbild sehen Sie die Mondbasis TI 99/4 A. Beginnen Sie das Spiel, indem Sie den Aktionsknopf des Joystick 1 drücken. An Stelle der Mondbasis erscheint nun Ihr Moon Racer, und die vordere Grafikebene setzt sich in Bewegung. Sie befinden sich jetzt in ...

Zone 1 ist die harmloseste Zone der Patrouille. Sie fahren mit Ihrem Moon Racer einfach so vor sich hin und müssen die Ihnen in den Weg kommenden Krater und Geröllhaufen überspringen. Zum Springen drücken Sie den Joystick einfach nach unten.

Achten Sie jedoch darauf, daß Sie die Krater und Geröllhaufen exakt überspringen, da Sie sonst einen Racer verlieren und mit einem neuen Racer bei der Basis beginnen müssen.

Für jeden erfolgreich übersprungenen Geröllhaufen erhalten Sie 10 Punkte, welche mit dem aktuellen Level multipliziert werden.

Irgendwann im Spiel bleibt die vordere Grafikebene stehen, und es erklingt ein Tusch. Jetzt haben Sie Zone 1 bewältigt und gelangen sofort danach in ...

In *Zone 2* wimmelt es nur so von Mondechsen, welche Ihnen im Weg sitzen. Da die Mondechsen jedoch so groß sind, daß Sie mit Ihrem Racer nicht darüber hinwegspringen können, so müssen Sie diese Echsen eliminieren. Dazu drücken Sie den Aktionsknopf am Joystick. Sofort schleudern Sie damit einen Torpedo in Richtung Mondechse. Ihr Torpedo hat jedoch nur eine bestimmte Reichweite, so daß Sie die Mondechse erst aus einer bestimmten Entfernung vernichten können. Haben Sie die Echse getroffen, so werden Ihnen 5 Punkte gutgeschrieben, und die Echse verwandelt sich in einen Geröllhaufen, den Sie überspringen müssen, um weitere 10 Punkte zu erhalten. Auch in Zone 2 werden die Punkte mit dem augenblicklichen Level multipliziert. Nachdem Sie Zone 2 erfolgreich überstanden haben sollten, ertönt wieder der Tusch, die vordere Grafikebene bleibt stehen, und Sie gelangen in ...

In *Zone 3* werden Sie von fliegenden Mondkrabben angegriffen. Diese Mondkrabben müssen Sie vernichten, da sonst Sie vernichtet werden. Weiterhin ist Zone 3 ein ziemlich unwegiges Gelände, wo Sie laufend über Krater und Geröllhaufen springen müssen. Es kommt noch hinzu, daß Zone 3 die längste Strecke auf Ihrer Patrouille ist und schier nicht enden möchte. Mondkrabben können Sie nur im Sprung vernichten, indem Sie den Joystick nach unten bewegen und während des Sprunges den Aktionsknopf drücken, um einen Torpedo abzufeuern. Als Schwierigkeit kommt noch dazu, daß Sie bei jedem Sprung nur einen Schuß abfeuern können. Wenn dieser Schuß danebengeht, so sind Sie der Krabbe hoffnungslos ausgeliefert. Treffen Sie jedoch hingegen, so werden Ihnen 25mal Level-Punkte gutgeschrieben. Sollte es Ihnen wider Erwarten gelingen, Zone 3 zu überleben, so ertönt der bekannte Tusch, und die Grafik bleibt für kurze Zeit stehen.
In den ersten 3 Zonen bekommen Sie nach einem erfolgreichen Beenden der Zone 100mal Zone mal Level-Bonuspunkte gutgeschrieben.
Wenn Sie es bis hierher geschafft haben sollten, so gelangen Sie in ...

Zone 4 ist radioaktiv verseucht, so daß Sie mit Ihrem Racer diese Zone nicht durchfahren können. Aus diesem Grund verwandelt sich Ihr Racer kurzum in einen Moon Jet, mit welchem Sie Zone 4 durchfliegen müssen. Zone 4 teilt sich in 4 Sektoren auf: In Sektor 1 kommen Ihnen radioaktive Wolken entgegen, denen es auszuweichen gilt.
Dazu bewegen Sie Ihren Jet mit dem Joystick nach oben und unten. Passen Sie jedoch auf, daß Sie am Boden keine Bruchlandung machen und daß Sie nicht mit dem Gebirge kollidieren.
In den Sektoren 2, 3 und 4 kommen Ihnen Strahlen entgegen, welche von Sektor zu Sektor mehr werden. Wie bei Sektor 1 gilt es auch hier, den Strahlen auszuweichen.
Pro bestrittenem Sektor werden Ihnen 500mal Sektor mal Level-Punkte gutgeschrieben.
Sollte es Ihnen wider Erwarten dennoch gelingen, auch Zone 4 zu meistern, so hören Sie wieder Ihren Tusch und kehren zur Basis zurück. Sie sehen die Basis, und die Titelmelodie erklingt. Durch Druck auf den Aktionsknopf starten Sie den nächsten Level, welcher sich in der Länge der einzelnen Zonen vom vorherigen Level unterscheidet.
Mit jedem Verlust eines Racers bzw. eines Jets kehren Sie wieder zur Basis zurück, von woaus Sie durch Druck auf den Aktionsknopf eine neue Patrouille beim gleichen Level beginnen können.
Nachdem alle 5 Fahrzeuge zerstört wurden, erscheint wieder das Titelbild. Sie können jetzt mittels des Aktionsknopfes ein neues Spiel starten.
Moon Race ist ein äußerst interessantes Spiel, welches äußerste Konzentration erfordert und sehr lange Spielfreude beschert.
Das Spiel ist sehr schwer zu beherrschen, und es bedarf einiger Stunden (Tage, Wochen, Monate), bis es einem gelingt, einen kompletten Level durchzuspielen. Viele Spieler werden bereits an Zone 3 verzweifeln. Es sei hierzu jedoch bemerkt, daß es zu schaffen ist. Spätestens bei Zone 4 werden die meisten Spieler sagen, es gibt keine Möglichkeit, durchzukommen. Hierzu sei jedoch gesagt, daß es einen Weg gibt, aber nur einen. Diesen Weg zu finden, sei jedoch dem Spielfreak überlassen.
Mir bleibt hierzu nur zu bemerken, daß ich allen Spielern viel Spaß und viel Erfolg wünsche bei MOON RACE.

Anmerkung: Die <ALPHA LOCK>-Taste darf während des Spieles nicht gedrückt sein, da es sonst keinen Weg gibt, die Patrouille erfolgreich durchzuführen.

```
100 ! ************************
110 ! *  M O O N   R A C E  *
120 ! *------------------------*
130 ! * TI 99/4 A + Ex.BASIC *
140 ! * Joystick 1 ist aktiv *
150 ! *------------------------*
160 ! * c 1984 Tronicsoft V.B*
170 ! ************************
180 !
190 OPTION BASE 1
200 DIM B(6,6),CH(6)
210 DATA 195,261,329,391,523,659,164,220,261,329,440,523,110,130,174,220,261,349
220 DATA 174,220,293,349,440,587,207,293,329,415,587,659,195,246,349,391,493,698
230 DATA 112,116,76,72,68,64
240 RESTORE 210
250 FOR I=1 TO 6 :: FOR K=1 TO 6 :: READ B(K,I):: NEXT K :: NEXT I
260 FOR I=1 TO 6 :: READ CH(I):: NEXT I
270 CH1$="30303030373F7FFFFFFFEFC7C6C6C6C600000000E0E8FEFFFFFFF7E363636363"
280 CH2$="03E00F0003E000000000000000000000000000000000000000000000000000000"
290 CALL CLEAR :: CALL ASCII :: CALL MAGNIFY(4)
300 CALL COLOR(1,16,14,2,16,14,3,2,1,4,2,1,8,15,14,9,5,1,10,4,1)
310 CALL SCREEN(14):: CALL HCHAR(22,1,88,224):: CALL HCHAR(6,1,96,160):: CALL HC
HAR(11,1,108,352)
320 P=1 :: FOR I=25 TO 28 :: CALL SPRITE(#I,136,5,9,P):: P=P+63 :: NEXT I
330 P=32 :: FOR I=21 TO 24 :: CALL SPRITE(#I,140,14,41,P):: P=P+65 :: NEXT I
340 P=1 :: FOR I=17 TO 20 :: CALL SPRITE(#I,128,4,61,P):: P=P+65 :: NEXT I
350 CALL POSITION(#17,X,Y):: CALL LOCATE(#17,X,Y+6)
360 P=32 :: FOR I=13 TO 16 :: CALL SPRITE(#I,132,5,81,P):: P=P+64 :: NEXT I
370 CALL SPRITE(#11,120,14,137,255,#10,124,4,153,121)
380 FOR I=21 TO 28 :: CALL MOTION(#I,0,-5):: NEXT I
390 FOR I=17 TO 20 :: CALL MOTION(#I,0,-10):: NEXT I
400 FOR I=13 TO 16 :: CALL MOTION(#I,0,-10):: NEXT I
410 CALL CHAR(60,CH1$)
420 RC=5 :: MR=1 :: P=0 :: CALL SCORE(P):: CALL RACER(RC):: CALL ZONE(1)
430 FOR I=1 TO 10 :: CALL HCHAR(24,11+I,32+I):: NEXT I
440 CALL TITEL(B(,),5)
450 CALL DELSPRITE(#5):: CALL SPRITE(#1,104,15,137,49,#2,112,9,137,49):: STV=0
460 DISPLAY AT(24,2)SIZE(3):USING "###":MR
470 FOR LEVEL=1 TO 3 :: ON LEVEL GOSUB 1180,1210,1230
480 CALL ZONE(LEVEL)
490 FOR DELAY=1 TO 40+10*MR
500 FOR LOOP=1 TO 2
510 IF STV THEN 530
520 ON LEVEL GOSUB 860,840,870
530 CALL PATTERN(#2,CH(LOOP)):: CALL SOUND(-50,-(LOOP+5),0):: CALL JOYST(1,X,Y):
: IF Y=-4 THEN GOSUB 570
540 CALL COINC(ALL,ST):: IF ST THEN 730
550 ON LEVEL GOSUB 860,890,880
560 NEXT LOOP :: NEXT DELAY :: P=P+100*LEVEL :: CALL SCORE(P):: CALL ENDZONE ::
NEXT LEVEL :: STV=0 :: GOTO 930
570 CALL LOCATE(#1,121,49,#2,121,49):: CALL PATTERN(#2,112):: CALL SOUND(-50,-3,
0)
580 IF LEVEL=3 THEN 650
590 FOR I=1 TO 13 :: CALL COINC(ALL,ST):: IF ST THEN 730
600 CALL COINC(#1,#11,45,SKP)
610 NEXT I
620 CALL LOCATE(#1,137,49,#2,137,49):: CALL SOUND(-50,-5,0)
630 IF NOT SKP THEN 640 ELSE P=P+10*MR :: CALL SCORE(P)
640 SKP=0 :: RETURN
650 SW=0 :: FOR I=1 TO 10 :: CALL PATTERN(#4,CH(2*(I/2-INT(I/2))+3))
660 CALL KEY(1,T,S):: IF T=18 THEN GOSUB 690
670 CALL COINC(ALL,ST):: IF ST THEN 730
680 NEXT I :: GOTO 620
```

```
690 IF SW THEN RETURN
700 SW=1 :: CALL SPRITE(#3,80,2,105,81,0,120):: CALL SOUND(-50,110,0,-7,0):: CAL
L DISTANCE(#1,#4,WT):: IF WT>11000 THEN 720
710 CALL DELSPRITE(#4):: CALL DELSPRITE(#3):: STV=0 :: P=P+25*MR :: CALL SCORE(P
)
720 CALL DELSPRITE(#3):: RETURN
730 CALL PATTERN(#1,44):: CALL SOUND(-50,-5,0)
740 CALL DELSPRITE(#2):: CALL SOUND(-50,-6,0)
750 CALL COLOR(#1,11):: CALL SOUND(-20,-7,0)
760 CALL DELSPRITE(#1)
770 RC=RC-1 :: CALL RACER(RC)
780 CALL PATTERN(#11,120):: CALL COLOR(#11,14):: STV=0 :: CALL DELSPRITE(#4)
790 IF RC>0 THEN GOSUB 1180 :: CALL WAIT(B(,)):: GOTO 450
800 GOSUB 1180 :: CALL SPRITE(#11,120,14,137,255,0,0,#10,124,4,153,121,0,0)
810 CALL TITEL(B(,),1)
820 RC=5 :: MR=1 :: P=0 :: CALL SCORE(P):: CALL RACER(RC):: CALL ZONE(1)
830 GOTO 450
840 CALL DISTANCE(#1,#11,WT):: IF WT<9000 THEN RETURN
850 CALL PATTERN(#11,84):: CALL COLOR(#11,5):: STV=1 :: RETURN
860 RETURN
870 CALL SPRITE(#4,72,13,113,240,0,-18):: STV=1 :: RETURN
880 CALL PATTERN(#4,CH(LOOP+2)):: RETURN
890 CALL KEY(1,T,S):: IF T=18 THEN 900 ELSE RETURN
900 CALL SPRITE(#3,80,2,121,81,0,120):: CALL SOUND(-50,110,0,-7,0):: CALL DISTAN
CE(#1,#11,WT):: IF WT>9000 THEN 920
910 CALL PATTERN(#11,120):: CALL COLOR(#11,14):: CALL DELSPRITE(#3):: P=P+5*MR :
: CALL SCORE(P)
920 CALL DELSPRITE(#3):: STV=0 :: RETURN
930 GOSUB 1250 :: CALL CHAR(60,CH2$):: CALL ZONE(4)
940 FOR LEVEL=1 TO 4
950 ON LEVEL GOSUB 1050,1060,1070,1080
960 FOR DELAY=1 TO 30+10*MR
970 FOR LOOP=5 TO 6
980 CALL SOUND(-300,-7,0):: CALL PATTERN(#1,CH(LOOP))
990 CALL JOYST(1,X,Y):: CALL MOTION(#1,-2*Y,0)
1000 CALL COINC(ALL,ST):: IF ST THEN 1090
1010 CALL POSITION(#1,X,Y):: IF X>150 THEN 1090
1020 NEXT LOOP :: NEXT DELAY :: P=P+500*LEVEL :: CALL SCORE(P):: NEXT LEVEL
1030 FOR I=1 TO 4 :: CALL DELSPRITE(#I):: NEXT I :: CALL CHAR(60,CH1$):: CALL EN
DZONE :: GOSUB 1180 :: CALL WAIT(B(,))
1040 MR=MR+1 :: GOTO 450
1050 CALL SPRITE(#2,44,15,107,240,0,-18):: RETURN
1060 CALL PATTERN(#2,60):: RETURN
1070 CALL SPRITE(#3,60,9,130,240,0,-18):: RETURN
1080 CALL SPRITE(#4,60,8,117,240,0,-18):: RETURN
1090 CALL PATTERN(#1,44):: CALL SOUND(-50,-5,0)
1100 CALL DELSPRITE(#2):: CALL SOUND(-50,-6,0)
1110 CALL COLOR(#1,16):: CALL SOUND(-20,-7,0)
1120 CALL DELSPRITE(#1)
1130 RC=RC-1 :: CALL RACER(RC)
1140 CALL PATTERN(#11,120):: CALL COLOR(#11,14):: STV=0 :: CALL DELSPRITE(#3,#4)
1150 IF RC>0 THEN CALL CHAR(60,CH1$):: GOSUB 1180 :: CALL WAIT(B(,)):: GOTO 450
1160 GOSUB 1180 :: CALL SPRITE(#11,120,14,137,255,0,0,#10,124,4,153,121,0,0)
1170 CALL CHAR(60,CH1$):: GOTO 810
1180 CALL COLOR(9,5,1,10,4,1):: CALL COLOR(#25,5,#26,5,#27,5,#28,5):: CALL COLOR
(#13,5,#14,5,#15,5,#16,5)
1190 CALL DELSPRITE(#4):: CALL PATTERN(#11,120):: CALL COLOR(#11,14,#17,4,#18,4,
#19,4,#20,4)
1200 CALL SPRITE(#11,120,14,137,255,0,-15,#10,124,4,153,121,0,-15):: RETURN
1210 CALL COLOR(9,7,1,10,4,1):: CALL COLOR(#25,7,#26,7,#27,7,#28,7):: CALL COLOR
(#13,7,#14,7,#15,7,#16,7)
1220 CALL DELSPRITE(#10):: CALL SPRITE(#11,120,14,137,255,0,-15):: RETURN
1230 CALL COLOR(9,7,1,10,8,1):: CALL COLOR(#17,8,#18,8,#19,8,#20,8)
```

```
1240 CALL SPRITE(#11,120,14,137,255,0,-15,#10,124,8,153,129,0,-15):: STV=0 :: RE
TURN
1250 CALL COLOR(9,16,1,10,5,1):: CALL COLOR(#25,16,#26,16,#27,16,#28,16):: CALL
COLOR(#13,16,#14,16,#15,16,#16,16)
1260 CALL DELSPRITE(#4):: CALL DELSPRITE(#2):: CALL PATTERN(#11,120):: CALL COLO
R(#11,14,#17,5,#18,5,#19,5,#20,5)
1270 CALL SPRITE(#11,120,14,137,255,0,-15,#10,120,14,137,121,0,-15,#1,64,11,114,
49):: RETURN
1280 SUB MOONRIVER(B(,),KY)
1290 DATA 1,1,6,783,30000,1,2,2,987,1174,4,880,1046,1,3,3,783,987,1,698,880,1,65
9,783
1300 DATA 1,587,698,1,1,4,659,783,2,523,30000,1,3,3,783,987,1,698,880,1,659,783,
1,587,698
1310 DATA 1,1,4,659,783,2,523,30000,2,4,10,587,30000,2,659,30000,1,2,6,523,30000
1320 DATA 1,1,4,659,30000,2,783,30000,1,2,6,1046,30000,1,3,4,1174,30000,2,1046,3
0000,1,1,6,783,30000
1330 DATA 1,3,2,783,30000,1,783,987,1,698,880,1,659,783,1,587,698,1,1,6,659,783,
1,3,1,659,783
1340 DATA 1,523,30000,1,783,987,1,698,880,1,654,783,1,587,698,1,1,6,659,783,1,2,
6,440,523
1350 DATA 1,4,2,440,698,4,349,587,1,6,4,349,587,2,391,659,2,1,12,329,523,0
1360 RESTORE 1290
1370 AZ,STV=0
1380 READ K :: IF K=0 THEN 1360
1390 READ INR
1400 FOR I=1 TO K
1410 FOR J=1 TO 6
1420 STV=STV+1 :: IF STV>AZ THEN READ AZ,M1,M2 :: STV=1
1430 CALL SOUND(400,M1,0,M2,0,B(J,INR),6):: CALL KEY(1,T,S):: IF S<>0 THEN SUBEX
IT
1440 NEXT J :: NEXT I
1450 GOTO 1380
1460 SUBEND
1470 SUB ENDZONE
1480 DATA 300,220,174,300,246,195,1200,261,220,600,293,246,300,329,261,300,349,2
93
1490 DATA 1200,391,329,300,329,261,300,349,293,600,391,329,600,440,349,600,493,2
93,1200,523,329
1500 CALL MOTION(#1,0,0,#2,0,0,#4,0,0,#3,0,0,#10,0,0,#11,0,0):: RESTORE 1480
1510 FOR I=1 TO 13 :: READ L,M1,M2 :: CALL SOUND(L,M1,0,M2,0):: NEXT I
1520 SUBEND
1530 SUB TITEL(B(,),KY)
1540 CALL SPRITE(#1,92,16,113,96,#2,100,16,113,128,#3,92,2,111,98,#4,100,2,111,1
30,#5,60,15,137,45)
1550 CALL MOONRIVER(B(,),KY)
1560 FOR I=1 TO 4 :: CALL DELSPRITE(#I):: NEXT I
1570 SUBEND
1580 SUB SCORE(P)
1590 DISPLAY AT(1,19):USING "mnc#######":P
1600 SUBEND
1610 SUB ZONE(Z)
1620 DISPLAY AT(1,1)SIZE(5):USING "abc #":Z
1630 SUBEND
1640 SUB RACER(RC)
1650 CALL HCHAR(1,12,32,5)
1660 IF RC=0 THEN SUBEXIT
1670 CALL HCHAR(1,12,89,RC)
1680 SUBEND
1690 SUB WAIT(B(,))
1700 CALL SPRITE(#11,120,14,137,255,0,0,#10,124,4,153,121,0,0,#5,60,15,137,49)
1710 CALL MOONRIVER(B(,),1)
1720 SUBEND
1730 SUB ASCII
```

```
1740 DATA ,0003050606050 3,00C4AC2424A4C4,00F79497F414F7,00D252D25F42C2,003F080A0
B0A0A,00FF00DE12121E
1750 DATA 00F500B5D59595,00CF08C80F00CF,00FE013DA5A5BD,00D010B810101,000010107C1
01,012140860801 03,200408011882100C
1760 DATA 0020C108C304003,8610400C600CC118,00182424242418,00081808080808,0018240
408103C,00182408042418,00282828380808
1770 DATA 003C2038042418,00182038242418,003C040810101,00182418242418,0018242 41C0
418,00003030003 03,00003030003 0102
1780 DATA 003E00F0003E,,,,0000000000F0909,0F0F,00000000000000E
1790 DATA FCFF,00000000008F4929,FF2F408,00000000000000E,FCFF,0102050F7FED8B1B,8B
ED7F0F050201
1800 DATA C020D0F8F8D8ED6E,ECD8F8F8D020C,0103050EFF3D0B1B,0B3DFF0E050301,C0E0D03
8F8D8EC6E,EDD8F838D0E0C,
1810 DATA 00000000009F7F9F,,0000000000C0E0C,0E173F7FCF0E0E4F,3F0F1F3F7F7F3F1F,00
00808,80C0C0EFFEFCF8F
1820 DATA ,007848F8FCFE44,,,1B24242020A04,1F2828180F0A291,00808788889067
1830 DATA 0080808E111131CF,FFFFFFFFFFFFFFFF,00F808136282FB,000000D65951D1,000000
74704074,00001CE3A2A21C,0000000E3050900F
1840 DATA 0000D824242522,000000384478C03C,0408143F22227F7F,FEFFFF,000000E02020F0
FC,7FFFFF,FFFFFFFFFFFFFFFF
1850 DATA 00F08087F414F7,000000BD2525BD,0000005D9C101D,,00000054927C9254,0000000
0010E01,0000002A493E492A
1860 DATA ,00000044AA92AA44,000001020508,0000002255495522,,01070D1F3F77FFFB,
1870 DATA 80F0F87CEEBEFFEF,FFFFFFFFFFFFFFFF,FF7F3F1F07,FFFFFFFFFFFFFFFF,FFFEFCF8
E,000103070F1F3F3F,7FFF
1880 DATA C0E0F0E0F8F8FCFE,FEFF,FF7FFF7F7F7F3F1E,1C0C,FFFEFCF8FCFCF8F,F06,000000
0101010307
1890 DATA 07070F1F3F7FFFFF,00008080C08080C,E0E0F0F8FCFCFEFF,FFFF7F3F1F070301,,FF
FFFEF8F0E0C08,,
1900 RESTORE 1740
1910 FOR I=32 TO 143 :: READ C$ :: CALL CHAR(I,C$):: NEXT I
1920 SUBEND
```

SLICKS

Ein Spiel mit toller Grafik

SLICKS ist auf dem TI 99/4 A in Extended BASIC geschrieben und wird mittels Joystick 1 gesteuert.

Autorennen gibt es viele.

Wir unterscheiden hierbei zwischen 2 Arten: Die erste Art simuliert ein Autorennen in der Draufsicht der Straße und hat eine schlichte Grafik. Solche Autorennen sind schon mehrfach als BASIC-Programme für die verschiedensten Computersysteme veröffentlicht worden.
Bei der zweiten Art handelt es sich um Autorennen, die das Gefühl, im Rennwagen zu sitzen, perfekt simulieren. Das Rennen läuft so ab, als ob man selbst am Steuer sitzt. Diese Autorennen bieten eine perfekte Grafik und einen ausgezeichneten 3D-Effekt. Leider handelt es sich hierbei um Maschinenprogramme, die meist für teures Geld auf Moduln zu kaufen sind. Dies alles ändert sich jedoch schlagartig mit diesem Programm, denn hier kommt SLICKS, das Autorennen der zweiten Art, als BASIC-Programm.
Tippen Sie das Programm ein, und nehmen Sie Platz in Ihrem TI Formel 1-Rennwagen. Geben Sie den Befehl

RUN, und in ca. einer Minute wird das Rennen beginnen.
Nachdem das Titelbild erschienen ist und die Melodie erklingt, drücken Sie den roten Aktionsknopf am linken Joystick, und das Rennen beginnt. Steuern Sie Ihren gelben Rennwagen mittels des Joysticks nach rechts oder links. Weichen Sie hierbei allen möglichen Sachen, die Ihnen entgegenkommen, aus. Die Geschwindigkeit steigert sich in 11 Stufen, die an der Höhe des Motorengeräusches zu unterscheiden sind. Haben Sie einen Gegenstand erforderlich umfahren, so werden Ihnen hierfür Punkte gutgeschrieben.
Wenn Sie jedoch einen Unfall bauen oder von der Straße abkommen, so wird Ihnen ein Auto abgezogen. Zu Spielbeginn stehen Ihnen 3 Fahrzeuge zur Verfügung, mit denen Sie das Rennen bestreiten.
Vorstehendes Bild zeigt eine Zusammenstellung der verschiedenen Situationen auf der Straße. Es können Ihnen also entgegenkommen:
Rote Rennwagen. Diese sind am einfachsten zu umfahren.
Grüne Ölflecken. Hier wird die Sache schon etwas schwieriger.
Blaue mit Wasser gefüllte Schlaglöcher, die am schwierigsten zu beherrschen sind.
Je nach Geschwindigkeit werden Ihnen an Punkten gutgeschrieben:

Rennwagen:	1 bis 11 Punkte
Ölflecken:	2 bis 22 Punkte
Schlaglöcher:	3 bis 33 Punkte

Nachdem Sie alle 3 Rennwagen in die „ewigen Jagdgründe" befördert haben, so erscheint wieder der Titelschriftzug, und die Melodie erklingt. Jetzt können Sie durch Druck auf den roten Aktionsknopf ein neues Rennen beginnen.
Beim Spielverlauf sind keine Besonderheiten zu beachten. Sogar die „ALPHA LOCK" Taste kann eingerastet sein.

```
100 !*********************
110 !*       SLICKS      *
120 !*-------------------*
130 !*     TI 99/4 A     *
140 !*  Extended BASIC   *
150 !*-------------------*
160 !* (c)1984 Tronicsoft *
170 !*       (V.B)       *
180 !*********************
190 !
200 OPTION BASE 1
210 DIM B(3,7),PZ(4),FK(4),ZH(4)
220 DATA 195,261,329,164,220,261,174,220,293,195,246,349,184,261,293,207,293,329
,174,220,261
230 RESTORE 220
240 FOR I=1 TO 7 :: FOR K=1 TO 3 :: READ B(K,I):: NEXT K :: NEXT I
250 CALL CLEAR :: CALL SCREEN(9):: CALL COLOR(0,9,9):: CALL HCHAR(1,1,30,768)
260 DATA 70,90,106,121,1,3,6,10,109,121,141,169
270 FOR I=1 TO 4 :: READ PZ(I):: NEXT I
280 FOR I=1 TO 4 :: READ FK(I):: NEXT I
290 FOR I=1 TO 4 :: READ ZH(I):: NEXT I :: RANDOMIZE
300 FOR I=1 TO 14 :: CALL COLOR(I,1,1):: NEXT I
310 CALL GRAFIK :: CALL BILD
320 CB$="DA828200DA0A0ADB0E0404000404046EDA8282008382820A5B5090001881415B"
330 CS$="0000381000000000001038100000000000107C380000000000000387CFEFE7C"
340 FOR I=12 TO 21 :: CALL HCHAR(12,I,I+36):: NEXT I
350 CALL COLOR(0,6,6,1,4,2,2,13,6,5,16,2,6,16,2,7,16,2,8,16,2,9,16,2,10,16,2,11,
16,2,12,16,2)
360 CALL COLOR(13,13,4,14,4,4,3,16,6,4,16,6):: CALL HCHAR(1,1,32,32):: CALL C2 :
: CALL TITEL(CB$,CS$,B(,)):: CALL RECHAR :: GOSUB 1000
370 CALL SPRITE(#1,135,11,175,121)
380 SPEED=11
```

Weiter Seite 46

Impressum: Computronic erscheint alle 2 Monate im Verlag: Tronic-Verlagsgesellschaft mbH, Landstr. 29, 3444 Wehretal 1, Tel.: (0 56 51) 4 06 43 / 4 06 93. Redaktion: (Verantw.) Axel Gredé, Rainer Beck, Frank Brall. Freie Mitarbeiter: Volker Becker, Rolf Freitag, H. Grede, Gesamtherstellung: D+V Druckhaus Dierichs Kassel. Vertrieb: Inland (Groß-, Einzel- und Bahnhofsbuchhandel), sowie Österreich und Schweiz: Verlagsunion, 6200 Wiesbaden. Anzeigenverwaltung u. Software-Service: A. Kratzenberg. Anzeigenpreis: Es gilt die Anzeigenliste Nr. 1. Bitte Media-Unterlagen anfordern. Bezugspreise: Einzelheft (Inland) DM 6,50, Abonnement (Inland) DM 35,–, (Ausland) DM 45,–. Autoren und Manuskripte: Bei Zusendung von Manuskripten und Datenträger erteilt der Autor dem Verlag die Genehmigung zum Abdruck und Versand der veröffentlichten Programme auf Datenträger. Für die mit Namen des Verfassers gekennzeichneten Beiträge übernimmt die Redaktion lediglich die presserechtliche Verantwortung. Alle in dieser Zeitschrift veröffentlichten Beiträge sind urheberrechtlich geschützt. Nachdruck (auch auszugsweise) und Vervielfältigung nur mit ausdrücklicher Genehmigung des Verlages. Für unaufgeforderte Einsendungen von Manuskripten, Tonträgern und Software wird keine Haftung übernommen.

```
390 V=(RND*2*SPEED-SPEED)
400 CALL C2
410 CALL SOUND(-4250,10*SPEED,0,-6,5)
420 CALL SPRITE(#2,132,9,99,121,SPEED,V)
430 FOR I=3 TO 4
440 CALL JOYST(1,X,Y):: CALL MOTION(#1,0,3*X):: CALL C1 :: CALL PAUSE(42-2*SPEED
)
450 CALL POSITION(#1,Y,X):: IF (X>185)+(X<57)THEN 730
460 CALL POSITION(#2,X,Y):: IF X>=PZ(I)THEN CALL PATTERN(#2,131+I)
470 CALL JOYST(1,X,Y):: CALL MOTION(#1,0,3*X):: CALL C2 :: CALL PAUSE(42-2*SPEED
)
480 CALL POSITION(#2,X,Y):: IF X>=PZ(I)THEN CALL PATTERN(#2,131+I):: GOTO 500
490 GOTO 440
500 NEXT I
510 P=P+(SPEED-10)
520 CALL PEEK(-31877,CN):: IF (CN AND 32)=32 THEN 730
530 CALL C1 :: CALL DELSPRITE(#2):: CALL JOYST(1,X,Y):: CALL MOTION(#1,0,3*X)
540 CALL PUNKTE(P)
550 SPEED=SPEED+1 :: IF SPEED>21 THEN SPEED=21
560 ON INT(RND*2.5)+1 GOTO 390,570,840
570 V=(RND*128)-64 :: SH=V/10
580 CALL C2
590 CALL SOUND(-4250,10*SPEED,0,-6,5)
600 FOR I=1 TO 3 STEP 2
610 CALL JOYST(1,X,Y):: CALL MOTION(#1,0,3*X):: CALL C1
620 CALL POSITION(#1,Y,X):: IF (X>185)+(X<57)THEN 730
630 CALL SPRITE(#2,139+I,8,ZH(I),121+FK(I)*SH)
640 CALL JOYST(1,X,Y):: CALL MOTION(#1,0,3*X):: CALL C2
650 CALL SPRITE(#2,140+I,8,ZH(I+1),121+FK(I+1)*SH)
660 NEXT I
670 P=P+3*(SPEED-10)
680 CALL PEEK(-31877,CN):: IF (CN AND 32)=32 THEN 730
690 CALL C1 :: CALL DELSPRITE(#2):: CALL JOYST(1,X,Y):: CALL MOTION(#1,0,3*X)
700 CALL PUNKTE(P)
710 SPEED=SPEED+1 :: IF SPEED>21 THEN SPEED=21
720 GOTO 390
730 CALL MOTION(#1,0,0):: CALL DELSPRITE(#2):: CALL COLOR(#1,10)
740 FOR I=0 TO 15 STEP 2 :: CALL SOUND(-100,-7,I)
750 ON I+1 GOSUB 800,800,800,800,810,810,810,810,820,820,820,820,830,830,830,830
760 NEXT I
770 CALL DELSPRITE(#1)
780 ATO=ATO-1 :: CALL AUTO(ATO):: IF ATO=0 THEN 1010
790 GOTO 370
800 CALL COLOR(#1,10):: CALL PATTERN(#1,33):: RETURN
810 CALL COLOR(#1,11):: CALL PATTERN(#1,34):: RETURN
820 CALL COLOR(#1,12):: CALL PATTERN(#1,38):: RETURN
830 CALL COLOR(#1,15):: CALL PATTERN(#1,39):: RETURN
840 V=(RND*128)-64 :: SH=V/10
850 CALL C2
860 CALL SOUND(-4250,10*SPEED,0,-6,5)
870 FOR I=1 TO 3 STEP 2
880 CALL JOYST(1,X,Y):: CALL MOTION(#1,0,3*X):: CALL C1
890 CALL POSITION(#1,Y,X):: IF (X>185)+(X<57)THEN 730
900 CALL SPRITE(#2,123+I,3,ZH(I),121+FK(I)*SH)
910 CALL JOYST(1,X,Y):: CALL MOTION(#1,0,3*X):: CALL C2
920 CALL SPRITE(#2,124+I,3,ZH(I+1),121+FK(I+1)*SH)
930 NEXT I
940 P=P+2*(SPEED-10)
950 CALL PEEK(-31877,CN):: IF (CN AND 32)=32 THEN 730
960 CALL C1 :: CALL DELSPRITE(#2):: CALL JOYST(1,X,Y):: CALL MOTION(#1,0,3*X)
970 CALL PUNKTE(P)
980 SPEED=SPEED+1 :: IF SPEED>21 THEN SPEED=21
990 GOTO 390
```

```
1000 P=0 :: ATO=3 :: CALL PUNKTE(P):: CALL AUTO(ATO):: RETURN
1010 CALL TITEL(CB$,CS$,B(,))
1020 GOSUB 1000
1030 GOTO 370
1040 SUB POEME(B(,),KY)
1050 DATA 2,1,3,311,30000,3,329,30000,2,2,6,440,30000,1,3,1,293,30000,1,329,3000
0,1,391,30000
1060 DATA 1,4,1,349,30000,1,329,30000,1,311,30000,2,1,1,349,30000,5,329,30000,1,
5,3,369,30000,1,1,3,391,30000
1070 DATA 2,6,5,523,30000,1,493,30000,1,2,1,415,30000,1,440,30000,1,659,523
1080 DATA 1,5,2,587,493,1,523,440,2,4,1,523,440,5,493,391
1090 DATA 1,1,3,659,523,1,2,3,659,523,1,7,1,659,523,1,587,493,1,493,415,1,2,1,52
3,440
1100 DATA 1,587,493,1,659,523,2,3,6,880,698,2,4,1,880,698,1,783,659,2,1396,880,1
,1318,783,1,1174,698
1110 DATA 1,2,3,1046,659,1,7,3,1046,587,2,1,1,1046,659,1,987,30000,1,659,523
1120 DATA 3,783,659,2,4,1,783,30000,1,587,30000,1,783,30000,1,698,30000,1,659,30
000,1,622,30000
1130 DATA 2,1,1,698,30000,5,659,30000
1140 DATA 1,1,3,659,523,1,2,3,659,523,1,7,1,659,523,1,587,493,1,493,415,1,2,1,52
3,440
1150 DATA 1,587,493,1,659,523,2,3,6,880,698,2,4,1,880,698,1,783,659,2,1396,880,1
,1318,783,1,1174,698
1160 DATA 1,2,3,1046,659,1,7,3,1046,587,2,1,1,1046,659,1,987,30000,1,659,523
1170 DATA 3,783,659,2,4,1,783,30000,1,587,30000,1,783,30000,1,698,30000,1,659,30
000,1,622,30000
1180 DATA 2,1,1,698,30000,5,659,30000,0,0
1190 RESTORE 1050
1200 AZ,STV,CL=0
1210 READ K :: IF K=0 THEN 1190
1220 READ INR
1230 FOR I=1 TO K
1240 FOR J=1 TO 3
1250 CALL C1
1260 STV=STV+1 :: IF STV>AZ THEN READ AZ,M1,M2 :: STV=1
1270 CALL SOUND(420,M1,0,M2,0,B(J,INR),6):: CALL KEY(1,T,S):: IF S<>0 THEN SUBEX
IT
1280 CALL C2
1290 NEXT J :: NEXT I
1300 GOTO 1210
1310 SUBEND
1320 SUB RECHAR
1330 DATA 00182424242418,00081808080808,00182404081030,00182408042418,0028282838
0808,003C2038042418,00182038242418
1340 DATA 003C040810101,00182418242418,001824241C0418
1350 RESTORE 1330
1360 CALL HCHAR(12,12,30,10):: CALL COLOR(3,11,2,4,11,2)
1370 FOR I=48 TO 57 :: READ C$ :: CALL CHAR(I,C$):: NEXT I
1380 SUBEND
1390 SUB PUNKTE(P)
1400 DISPLAY AT(1,23)SIZE(6):USING "######":P
1410 SUBEND
1420 SUB AUTO(ATO)
1430 CALL HCHAR(1,3,32,3)
1440 IF ATO=0 THEN SUBEXIT
1450 CALL HCHAR(1,3,59,ATO)
1460 SUBEND
1470 SUB TITEL(C1$,C2$,B(,))
1480 CALL CHAR(124,C1$)
1490 PS=97 :: FOR I=1 TO 4 :: CALL SPRITE(#I,123+I,10,73,PS):: PS=PS+16 :: NEXT
I
1500 CALL POEME(B(,),1)
1510 FOR I=1 TO 4 :: CALL DELSPRITE(#I):: NEXT I
```

```
1520 CALL CHAR(124,C2$)
1530 SUBEND
1540 SUB C1 :: CALL COLOR(7,16,2,8,16,2,11,16,2,5,2,2,6,2,2,9,2,2,10,2,2,12,2,2,
13,13,4,14,4,4):: SUBEND
1550 SUB C2 :: CALL COLOR(5,16,2,6,16,2,9,16,2,10,16,2,12,16,2,7,2,2,8,2,2,11,2,
2,14,13,4,13,4,4):: SUBEND
1560 SUB PAUSE(Z):: FOR I=1 TO Z :: NEXT I :: SUBEND
1570 SUB BILD
1580 DATA (*-/)+(*(*-/)+....)+(*-/)+-/(*)+
1590 DATA ##############$Vvw^%##############,##############$`axzed%############,####
#######$bcWy(W9f%############
1600 DATA ##########$PQWWPsWWYX%##########,#########$RSWWWqtWWWEZ%#########,####
####$TSWWWWrqWWWWE\%########
1610 DATA #######$@AWWWWWhlWWWWIH%#######,######$BCWWWWWWimWWWWWWKJ%######,####
##$DEWWWWWWWjnWWWWWWWWML%#####
1620 DATA #####$FGWWWWWWWWkoWWWWWWWWWON%####
1630 DATA 15,13,128,15,20,129,16,11,136,16,22,136,18,9,130,18,24,130
1640 DATA 19,9,131,19,24,131,22,5,138,23,5,139,22,28,138,23,28,139
1650 DATA 7,4,63,3,7,61,5,10,60,5,12,62,6,19,61,4,23,60,4,25,62,7,29,63
1660 RESTORE 1580
1670 FOR Z=14 TO 24 :: READ C$ :: FOR S=1 TO 32
1680 CALL VCHAR(Z,S,ASC(SEG$(C$,S,1)))
1690 NEXT S :: NEXT Z
1700 FOR I=1 TO 12 :: READ Z,S,C :: CALL VCHAR(Z,S,C):: NEXT I
1710 CALL MAGNIFY(2)
1720 FOR I=21 TO 28 :: READ Z,S,C :: CALL SPRITE(#I,C,16,8*Z,8*S):: NEXT I
1730 SUBEND
1740 SUB GRAFIK
1750 DATA ,422418BDFF7E3C5A,422418ADFD563C4A,FFFFFFFFFFFFFFFF,FFFEFCF8F0E0C08,FF
7F3F1F0F070301,022010A93D502842
1760 DATA 0220008821100042,081C3C7F7FFFFFFF,03071F1F3F7F7FFF,4070F8FCFEFFFFFF,C0
E0F8F8FCFEFEFF,,0000000C1F7F7FFF
1770 DATA ,0000000000C0F8FE,0003050606050 3,00C4AC2424A4C4,00F79497F414F7,00D252D
25F42C2,003F080A0B0A0A
1780 DATA 00FF00DE12121E,00F500B5D59595,00CF08C80F00CF,00FE013DA5A5BD,00D010B810
101,0000303000303,0010387CBAFE92
1790 DATA 010F3FFFFF7F0C,00187C7EFF7F3E1,0080C0ECFE7F0E04,00007C3E1F0E04,0103070
F1F3E7EFC,F0E0C08,0103070F1F3F7EFC
1800 DATA F8F0E0C08,0103070F1F3F7FFE,FCF8F0E0C08,0103070F1F3F7FFF,FCF8F8F0E0C08,
80C0E0F0F87C7E3F,0F070301
1810 DATA 80C0E0F0F8FC7E3F,1F0F070301,80C0E0F0F8FCFE7F,3F1F0F070301,80C0E0F0F8FC
FEFF,3F1F1F0F070301,0103070E1C3878F
1820 DATA C08,0103070F1E3C78F,E0C08,0103070F1E3E7CF8,0044444444444438,0102040C18
3060C,
1830 DATA 80C0E070381C1E0F,0301,80C0E0F0783C1E0F,070301,80C0E0F0787C3E1F,0038080
808080838,80402030180C0603
1840 DATA 0000000000000007C,0103060C183870E,8,0103070E1C3870E,C08,80C06030181C0E0
7,01
1850 DATA 80C0E070381C0E07,0301,1F1F1F1F1F1F1F1F,1F1F1F1F3F3F3F3F,3F3F3F3F7F7F7F
7F,7F7F7F7FFFFFFFFF,F8F8F8F8F8F8F8F8
1860 DATA F8F8F8F8FCFCFCFC,FCFCFCFCFEFEFEFE,FEFEFEFEFFFFFFFF,0303030307070707,07
07070707070F0F,0F0F0F0F0F0F0F0F,C0C0C0C0E0E0E0E
1870 DATA E0E0E0E0E0E0F0F,F0F0F0F0F0F0F0F0F,0000000101010101,0000008080808 08,01010
10101010101,0303030303030303,8080808080808 08
1880 DATA C0C0C0C0C0C0C0C,0010101000101 01,0030080804080 83,0000205408,,0000103810
381,0000081C081C08
1890 DATA 00000018183C183C,7E183C181818,000018183C18,0000183C183C3C,00183C3C5A7E
5A,185A7E5A18BDFF99,00183C183C7E1818
1900 DATA ,183C7E18183C7EFF,18183C7EFF181818,0000081C08,00040E1F0E04,08142E5F2E1
408,2854AE5BAE5428
1910 RESTORE 1750
1920 FOR I=32 TO 143 :: READ C$ :: CALL CHAR(I,C$):: NEXT I
1930 SUBEND
```

„Panzerschlacht"

Als Panzerkommandant müssen Sie versuchen, möglichst viele der feindlichen Hubschrauber abzuschießen, die laufend über das Gelände im Tiefflug hinwegziehen. Aber Vorsicht! Ihr Gegenspieler, der die gleiche Aufgabe hat und ebenfalls einen Panzer steuert, wird versuchen, Ihren Panzer abzuschießen. In der Zeit, bis Ihnen von der Eisenbahn ein neuer Panzer geliefert wird, kann er dann nämlich ungehindert Ihre Hubschrauber abschießen.

Davor können Sie sich schützen, indem Sie sich im Wald unter Bäumen verstecken. Eine perfekte Tarnung ist das aber nicht! Spätestens bei der nächsten Bewegung des Panzers verraten Sie Ihre Position.

Beide Spieler bekommen je abgeschossenen Feindhubschrauber 5 Punkte, für abgeschossene Panzer gibt's nur 2 Punkte. Schüsse auf die eigenen Hubschrauber (erkenntlich an der gleichen Farbe) haben keinerlei Folgen. Die Panzer sind schwieriger zu treffen als die Hubschrauber. Der Nachschub an Panzern und Hubschraubern ist unbegrenzt. Beide Panzer können nicht gleichzeitig feuern, sondern immer nur abwechselnd. Während der Anlieferung eines neuen Panzers durch die Eisenbahn, welche sich mit einer Dampfpfeife ankündigt und auch verabschiedet, herrscht Waffenstillstand, d. h., die Steuerung für beide Panzer ist unterbrochen. Nach der Lieferung werden beide Panzer in Ausgangsstellung gebracht und das Spiel geht weiter. Dies ist auch daran zu erkennen, daß sich die Rotorblätter der Hubschrauber wieder drehen.

Wer als erster 100 Punkte erreicht, wird – begleitet von einem Fanfarenstoß – zum Sieger erklärt.

Das Hauptprogramm besteht lediglich aus der Joystickabfrage und der Bewegungssteuerung für die beiden Panzer einschließlich Berechnung des jeweils richtigen Panzer-Charakters für die vier Bewegungsrichtungen. Da nicht beide Panzer gleichzeitig kontrolliert werden können, sondern abwechselnd immer nur einer, mußte ich darauf achten, daß die Abarbeitungszeit des Hauptprogramms so kurz wie möglich bleibt. Daher ist die Anzahl der Rechenvorgänge und der Variablen sowie deren Länge möglichst klein gehalten. Die Möglichkeit, Multiple-Statement-Zeilen zu bilden, wurde weitgehend ausgenutzt.

Dies gilt größtenteils auch für die Subroutinen, z. B. für Erzeugung und Kontrolle der Granate nach Betätigung der Feuertaste.

Durch die relative Schnelligkeit des Hauptprogramms entsteht der Eindruck, daß die beiden Panzer gleichzeitig und unabhängig voneinander bewegt werden. Beide Panzer können jedoch nicht gleichzeitig schießen. Sobald eine Granate abgefeuert wurde, ist die Steuerung beider Panzer unterbrochen, bis ein Treffer erzielt ist oder die Granate den Bildschirmrand erreicht hat.

Hinweis zum Eintippen: Sollte die eine oder andere Zeile zu lang sein, um sie vollständig einzugeben, so ist sie nach vorläufiger Eingabe mit FCTN REDO zu wiederholen und danach zu vervollständigen.

```
100 ! ***********************
110 ! *   PANZERSCHLACHT    *
120 ! * ------------------- *
130 ! *ein actionspiel fuer*
140 ! * zwei joysticks in  *
150 ! * TI Extended Basic  *
160 ! * ------------------- *
170 ! *    (c) 1984 by      *
180 ! *   michael oswald    *
190 ! ***********************
200 CALL CLEAR :: CALL SCREEN(12)
210 CALL CHAR(96,"3F3F1EFE1E3F3F00",97,"C6FEFEFEFED61010",99,"1010D6FEFEFEFEC6",
100,"FCFC787F78FCFC00")
220 CALL TITEL(G$,B$)
230 CALL CHAR(102,"00280D34165C2400",103,"8801488421248822",104,"900824599C3A480
A",111,"0000001818000000")
240 CALL CHAR(115,"189050103B383890",116,"1812141088383812")
250 CALL CHAR(117,"0101010101010101")
260 CALL CHAR(64,"147C7EFFFE7FFE3C",72,"183CFE7FFFFE7C3C",80,"2418080808080000")
270 CALL CHAR(118,"080F1D0F1C0C1E0C",119,"0F1F0F0F0F0F1F0F")
280 CALL CHAR(128,"7E247E247E247E24",120,"0101010101010101",65,"F8F8F8F8",73,"F8
F8F8F8")
290 CALL CHAR(136,"FFFFFFFFFFFFFFFF",137,"01071F3F3F7F7F7F")
300 CALL CHAR(138,"C0F0F8FCFEFEFFFF",139,"FFFFFEFEFCFCE0C0",140,"7F1F1F1F0F07010
0")
310 CALL CHAR(105,"0000000000183C18",112,"0000183830181808")
320 CALL CHAR(107,"183C7F3E1C0C0808",108,"61F3F7722",109,"C48E24712")
330 CALL CHAR(141,"01FF031F3F1F150F",142,"E0F0F0FEFFFE54F8",143,"0000000040404040
0")
```

```
340 CALL CHAR(130,"387CFEFEFE929292",131,"6C7C285438")
350 CALL CHAR(132,"387CFEFEFE9E929E",133,"6C380044380")
360 CALL COLOR(5,7,1,6,13,1,7,2,1,12,2,10,13,10,16,14,6,1)
370 B(4)=0 :: B(8)=0
380 ! ***
390 ! *** AUFBAU GLEISANLAGEN
400 CALL VCHAR(16,6,128,9):: CALL VCHAR(16,27,128,9)
410 CALL HCHAR(15,7,73,1):: CALL HCHAR(15,6,120,1):: CALL HCHAR(15,28,65,1):: CA
LL HCHAR(15,27,120,1)
420 ! ***
430 ! *** AUFBAU DES WALDES
440 FOR CN=13 TO 19 STEP 2 :: FOR RN=1 TO 7 STEP 2 :: CALL VCHAR(RN,CN,64,1):: N
EXT RN
450 FOR RM=2 TO 20 STEP 2 :: CALL VCHAR(RM,CN,72,1):: NEXT RM
460 CALL VCHAR(9,CN,80,1)
470 FOR RO=11 TO 23 STEP 2 :: CALL VCHAR(RO,CN,64,1):: NEXT RO
480 CALL VCHAR(21,CN,80,1):: CALL VCHAR(22,CN,72,1):: CALL VCHAR(24,CN,80,1):: N
EXT CN
490 FOR CM=14 TO 20 STEP 2 :: CALL VCHAR(1,CM,72,1):: CALL VCHAR(2,CM,80,1)
500 FOR RP=3 TO 15 STEP 2 :: CALL VCHAR(RP,CM,72,1):: NEXT RP
510 FOR RQ=4 TO 16 STEP 2 :: CALL VCHAR(RQ,CM,64,1):: NEXT RQ
520 CALL VCHAR(17,CM,80,1):: FOR RR=18 TO 24 STEP 2 :: CALL VCHAR(RR,CM,64,1)::
NEXT RR
530 FOR RS=19 TO 23 STEP 2 :: CALL VCHAR(RS,CM,72,1):: NEXT RS :: NEXT CM
540 FOR RA=4 TO 20 STEP 4 :: CALL VCHAR(RA,12,64,1):: CALL VCHAR(RA+1,12,80,1)::
 CALL VCHAR(RA,21,72,1):: CALL VCHAR(RA+1,21,80,1):: NEXT RA
550 ! ***
560 ! *** AUFBAU DER TEICHE
570 DATA 5,5,137,5,7,138,7,7,139,7,5,140,5,25,137,5,26,138
580 DATA 6,26,139,6,25,140,7,27,137,7,28,138,8,28,139,8,27,140
590 RESTORE 570
600 FOR TEICH=1 TO 12 :: READ TR,TC,TP
610 CALL HCHAR(TR,TC,TP):: NEXT TEICH
620 CALL HCHAR(6,5,136,3):: CALL VCHAR(5,6,136,3)
630 ! ***
640 ! *** AUFBAU GEBUESCH
650 DATA 8,7,80,6,27,72,7,26,80,6,8,72,8,6,64
660 DATA 6,24,64,5,9,64,6,10,72,7,23,72,6,22,64,9,22,64,10,9,72
670 DATA 12,23,72,13,11,64,3,26,64,2,23,72,4,7,64,2,11,72
680 RESTORE 650
690 FOR BUSCH=1 TO 18 :: READ BR,BC,BP
700 CALL HCHAR(BR,BC,BP):: NEXT BUSCH
710 CALL SPRITE(#20,64,13,49,49,#21,64,7,41,201)
720 ! ***
730 ! *** AUFRUF PANZER UND HUBSCHRAUBER
740 CALL SPRITE(#4,99,7,96,208,#8,99,13,96,40):: P=1 :: N=1
750 CALL SPRITE(#1,115,13,1,1,10,6,#2,115,7,1,256,10,-6)
760 ! ***
770 ! *** HAUPTPROGRAMM
780 K=4 :: CALL JOYST(1,X,Y):: IF X+Y=0 THEN 800 :: IF X*Y THEN 780
790 J=10*X+Y :: P=(J=-4)-(J=4)+2*(J=-40)-2*(J=40)
800 CALL PATTERN(#4,98+P,#1,115,#2,116):: CALL MOTION(#4,-Y,X)
810 CALL KEY(1,Q,S):: IF S THEN GOSUB 890
820 K=8 :: CALL JOYST(2,W,Z):: IF W+Z=0 THEN 840 :: IF W*Z THEN 820
830 J=10*W+Z :: N=(J=-4)-(J=4)+2*(J=-40)-2*(J=40)
840 CALL PATTERN(#8,98+N,#1,116,#2,115):: CALL MOTION(#8,-Z,W)
850 CALL KEY(2,Q,U):: IF U THEN GOSUB 920
860 GOTO 780
870 ! ***
880 ! *** SUB PANZER FEUERT
890 CALL POSITION(#4,R,C):: CALL SOUND(100,-6,2)
900 A=ABS(P):: IF A=1 THEN CALL SPRITE(#3,111,7,R,C,-126*P,0)ELSE CALL SPRITE(#3
,111,7,R,C,0,63*P)
```

```
910 CALL COINC(#8,#3,12,T):: CALL COINC(#3,#1,16,S):: IF S OR T THEN 970 :: CALL
 POSITION(#3,M(1),M(2)):: IF M(A)>60 AND M(A)<196 THEN 910 ELSE CALL DELSPRITE(#
3):: RETURN
920 CALL POSITION(#8,R,C):: CALL SOUND(100,-6,2)
930 A=ABS(N):: IF A=1 THEN CALL SPRITE(#7,111,13,R,C,-126*N,0)ELSE CALL SPRITE(#
7,111,13,R,C,0,63*N)
940 CALL COINC(#4,#7,12,T):: CALL COINC(#7,#2,16,S):: IF S OR T THEN 970 :: CALL
 POSITION(#7,M(1),M(2)):: IF M(A)>60 AND M(A)<196 THEN 940 ELSE CALL DELSPRITE(#
7):: RETURN
950 ! ***
960 ! *** SUB FEINDHUBSCHRAUBER GETROFFEN
970 IF T THEN 1020 ELSE CALL DELSPRITE(#K-1):: CALL MOTION(#K/4,0,0):: CALL SOUN
D(-500,-6,2):: CALL PATTERN(#K/4,103)
980 CALL LOCATE(#K/4,3,(K-2)^2*7):: CALL PATTERN(#K/4,115):: CALL MOTION(#K/4,10
,(6-K)*3)
990 B(K)=B(K)+5 :: IF B(K)>=100 THEN 1380 ELSE DISPLAY AT(1,(32/K)^2/2-7)SIZE(3)
:B(K):: RETURN
1000 ! ***
1010 ! *** SUB FEINDPANZER GETROFFEN
1020 CALL DELSPRITE(#K-1):: CALL MOTION(#32/K,0,0,#K,0,0)
1030 CALL SOUND(3000,-6,0)
1040 FOR L=1 TO 5 :: CALL PATTERN(#32/K,102):: CALL COLOR(#32/K,16):: FOR L1=1 T
O 30 :: NEXT L1 :: CALL PATTERN(#32/K,103):: CALL COLOR(#32/K,2):: FOR L2=1 TO 3
0 :: NEXT L2 :: NEXT L
1050 FOR PS=1 TO 100 :: NEXT PS :: CALL PATTERN(#32/K,104):: FOR J=0 TO 28 STEP
2 :: CALL SOUND(-50,110,J,117,J,139,J,-6,J):: NEXT J
1060 CALL POSITION(#32/K,ROW,COL):: CALL SPRITE(#5,105,16,ROW-10,COL):: FOR L3=1
 TO 100 :: NEXT L3
1070 CALL PATTERN(#5,112):: FOR L4=1 TO 100 :: NEXT L4 :: CALL PATTERN(#5,107)
1080 FOR L5=1 TO 100 :: NEXT L5 :: CALL PATTERN(#5,108):: FOR L6=1 TO 100 :: NEX
T L6
1090 CALL PATTERN(#5,109):: FOR L7=1 TO 200 :: NEXT L7 :: CALL DELSPRITE(#5)
1100 B(K)=B(K)+2 :: IF B(K)>=100 THEN 1380 ELSE DISPLAY AT(1,(32/K)^2/2-7)SIZE(3
):B(K)
1110 ! ***
1120 ! *** PAUSE BIS LIEFERUNG NEUEN FEINDPANZERS
1130 D=0 :: IF K=8 THEN P=N
1140 CALL JOYST(K/4,X,Y):: IF X+Y=0 THEN 1160 :: IF X*Y THEN 1140
1150 J=10*X+Y :: P=(J=-4)-(J=4)+2*(J=-40)-2*(J=40)
1160 CALL PATTERN(#K,98+P,#1,115,#2,116):: CALL MOTION(#K,-Y,X):: D=D+1 :: IF D>
=50 THEN GOTO 1260
1170 CALL KEY(K/4,Q,S):: IF S THEN 1180 ELSE CALL PATTERN(#1,116,#2,115):: GOTO
1140
1180 CALL POSITION(#K,R,C):: CALL SOUND(100,-6,2)
1190 A=ABS(P):: IF A=1 THEN CALL SPRITE(#3,111,7,R,C,-126*P,0)ELSE CALL SPRITE(#
3,111,7,R,C,0,63*P)
1200 CALL COINC(#3,#K/4,16,S):: IF S THEN 1210 :: CALL POSITION(#3,M(1),M(2)):: 
IF M(A)>60 AND M(A)<196 THEN 1200 ELSE CALL DELSPRITE(#3):: GOTO 1140
1210 CALL DELSPRITE(#3):: CALL MOTION(#K/4,0,0):: CALL SOUND(-500,-6,2):: CALL P
ATTERN(#K/4,103)
1220 CALL LOCATE(#K/4,3,(K-2)^2*7):: CALL PATTERN(#K/4,115):: CALL MOTION(#K/4,1
0,(6-K)*3)
1230 B(K)=B(K)+5 :: IF B(K)>=100 THEN 1380 ELSE DISPLAY AT(1,(32/K)^2/2-7)SIZE(3
):B(K):: GOTO 1140
1240 ! ***
1250 ! *** SUB LIEFERUNG NEUER PANZER
1260 CALL MOTION(#K,0,0)
1270 CALL SOUND(100,-6,6):: CALL SOUND(-1200,2000,4,4000,4,6000,4,-6,4):: CALL S
OUND(50,-6,10)
1280 IF K=4 THEN CO=13 ELSE CO=7
1290 IF K=4 THEN SE=40 ELSE SE=208
1300 CALL SPRITE(#32/K,99,CO,240,SE,#10,118,2,216,SE,#11,119,CO,208,SE)
```

```
1310 FOR F=216 TO 128 STEP -1 :: CALL LOCATE(#10,F,SE,#11,F-8,SE):: CALL SOUND(-
10,2200,20,220,14,110,12,-6,3):: NEXT F
1320 FOR PAUSE=1 TO 200 :: NEXT PAUSE
1330 CALL SOUND(100,-6,6):: CALL SOUND(-1200,2000,4,4000,4,6000,4,-6,4):: CALL S
OUND(50,-6,10)
1340 FOR TIME=1 TO 500 :: NEXT TIME
1350 FOR G=120 TO 96 STEP -1 :: CALL LOCATE(#32/K,G,SE):: NEXT G
1360 FOR H=128 TO 216 :: CALL LOCATE(#10,H,SE,#11,H-8,SE):: CALL SOUND(-10,2200,
20,220,14,110,12,-6,3):: NEXT H
1370 GOTO 740
1380 CALL SCHLUSS(G$,B$,K)
1390 DISPLAY AT(23,4):"NOCH EINMAL ?  (j/n)" :: CALL KEY(0,K,S)
1400 IF NOT S THEN 1390 :: IF K=106 THEN CALL CLEAR :: GOTO 260 ELSE END
1410 ! ***
1420 ! *** TITELBILD UND SPIELREGELN
1430 SUB TITEL(G$,B$)
1440 CALL CHAR(136,"FFFFFFFFFFFFFFFF"):: CALL COLOR(14,6,1)
1450 CALL MAGNIFY(2):: CALL SPRITE(#20,100,13,96,24,#21,96,7,96,224)
1460 CALL HCHAR(5,10,136,14):: CALL HCHAR(7,10,136,14)
1470 DISPLAY AT(6,8):"PANZERSCHLACHT"
1480 DISPLAY AT(13,6):"ein spiel fuer zwei"
1490 DISPLAY AT(23,1):" (C) BY MICHAEL OSWALD 1984"
1500 FOR L=1 TO 90 :: CALL SOUND(-60,1400-(L*6),L/9):: NEXT L :: CALL SOUND(1500
,-6,0):: FOR L=0 TO 30 :: CALL SOUND(10,110,L,-7,L):: NEXT L
1510 FOR ZEIT=1 TO 500 :: NEXT ZEIT :: DISPLAY AT(19,6)BEEP:"LEERTASTE DRUECKEN!
"
1520 CALL KEY(0,K,S):: IF S=0 THEN 1520 ELSE GOTO 1530
1530 CALL DELSPRITE(#20,#21):: CALL MAGNIFY(1):: CALL CLEAR
1540 DISPLAY AT(13,1):"SPIELREGELN NOETIG ? (J/N)"
1550 CALL KEY(0,K,S):: IF S=0 THEN 1550 :: CALL CLEAR :: IF K=78 THEN 1630
1560 PRINT :"DIE SPIELER STEUERN JE EINENPANZER MITTELS JOYSTICKS."
1570 PRINT :"ES KOENNEN DIE FEINDLICHEN  HUBSCHRAUBER ABGESCHOSSEN   WERDEN(5 PU
NKTE), ABER AUCH DER JEWEILIGE FEINDPANZER(2 PUNKTE)."
1580 PRINT :"DIE BEIDEN PANZER KOENNEN   NICHT GLEICHZEITIG FEUERN."
1590 PRINT :"NACH ABSCHUSS EINES PANZERS KANN DER ANDERE NOCH KURZE  ZEIT WEITER
 FEUERN, BIS DEM GEGNER EIN NEUER PANZER GE- LIEFERT WIRD."
1600 PRINT :"WER ALS ERSTER 100 PUNKTE   HAT, IST SIEGER."
1610 PRINT :"  >LEERTASTE< FUER WEITER"
1620 CALL KEY(0,K,S):: IF S=0 THEN 1620 ELSE CALL CLEAR
1630 DISPLAY AT(1,1):"NAMEN DER OBERBEFEHLSHABER  ANGEBEN !" :: DISPLAY AT(8,1):
"FUER GRUEN :" :: DISPLAY AT(10,1):"GENERAL"
1640 DISPLAY AT(14,1):"FUER BRAUN :" :: DISPLAY AT(16,1):"GENERAL"
1650 ACCEPT AT(10,9)VALIDATE(UALPHA,NUMERIC)BEEP SIZE(24):G$
1660 ACCEPT AT(16,9)VALIDATE(UALPHA,NUMERIC)BEEP SIZE(24):B$
1670 DISPLAY AT(20,5):"ALPHA LOCK LOESEN !"
1680 DISPLAY AT(23,1):"  >LEERTASTE< FUER START" :: CALL KEY(0,K,S):: IF S=0 THE
N 1680 ELSE CALL CLEAR
1690 SUBEND
1700 ! ***
1710 ! *** SPIELENDE
1720 SUB SCHLUSS(G$,B$,K)
1730 CALL DELSPRITE(ALL):: CALL CLEAR :: CALL CHARSET
1740 CALL SOUND(100,400,0,1200,0,2400,0):: FOR L=0 TO 30
1750 CALL SOUND(-50,400,L,1200,L,2400,L):: NEXT L
1760 IF K=4 THEN SIEG$=B$ ELSE SIEG$=G$
1770 DISPLAY AT(2,1):"GENERAL ";SIEG$
1780 DISPLAY AT(4,1):"IST DER SIEGER !"
1790 FOR TON=1 TO 12 :: READ D,F
1800 CALL SOUND(D,F-3,0,F,0,F+3,0):: NEXT TON
1810 DATA 300,250,500,320,50,320,50,320,50,320,200,400,200,320,200,400,200,480,2
00,400,200,480,500,650
1820 RESTORE 1810
```

```
1830 IF K=4 THEN COL=7 ELSE COL=13
1840 CALL MAGNIFY(2)
1850 CALL SPRITE(#12,141,COL,18*8,23*8,#13,142,COL,18*8,24*8+8)
1860 FOR I=1 TO 500 :: NEXT I :: CALL SOUND(100,-4,0,110,5,220,5,3000,3)
1870 CALL SPRITE(#14,143,COL,16*8,25*8+4):: FOR I=1 TO 100 :: NEXT I
1880 CALL SPRITE(#5,105,16,16*8,24*8+4,-3,0):: FOR L3=1 TO 100 :: NEXT L3
1890 CALL PATTERN(#5,112):: FOR L4=1 TO 100 :: NEXT L4 :: CALL PATTERN(#5,107)
1900 FOR L5=1 TO 100 :: NEXT L5 :: CALL PATTERN(#5,108):: FOR L6=1 TO 100 :: NEX
T L6
1910 FOR L5=1 TO 100 :: NEXT L5 :: CALL PATTERN(#5,108):: FOR L6=1 TO 100 :: NEX
T L6
1920 CALL MOTION(#5,0,0):: CALL POSITION(#5,DOR,DOC)::
1930 CALL SPRITE(#5,131,16,DOR,DOC,#6,130,16,DOR-16,DOC):: FOR T1=1 TO 200 :: NE
XT T1
1940 CALL PATTERN(#5,133):: FOR T2=1 TO 90 :: NEXT T2 :: CALL PATTERN(#6,132)::
FOR I=1 TO 50 :: NEXT I :: CALL PATTERN(#6,130)
1950 FOR T3=1 TO 90 :: NEXT T3 :: CALL PATTERN(#6,132):: FOR I=1 TO 50 :: NEXT I
 :: CALL PATTERN(#6,130)
1960 FOR T4=1 TO 90 :: NEXT T4 :: CALL PATTERN(#5,131):: FOR T5=1 TO 200 :: NEXT
 T5
1970 CALL DELSPRITE(#5,#6):: FOR I=1 TO 300 :: NEXT I
1980 CALL SOUND(100,-4,0,110,9,220,9,3000,3):: CALL DELSPRITE(#14)
1990 FOR REST=1 TO 750 :: NEXT REST
2000 CALL DELSPRITE(ALL):: CALL MAGNIFY(1)
2010 SUBEND
```

ZX-81

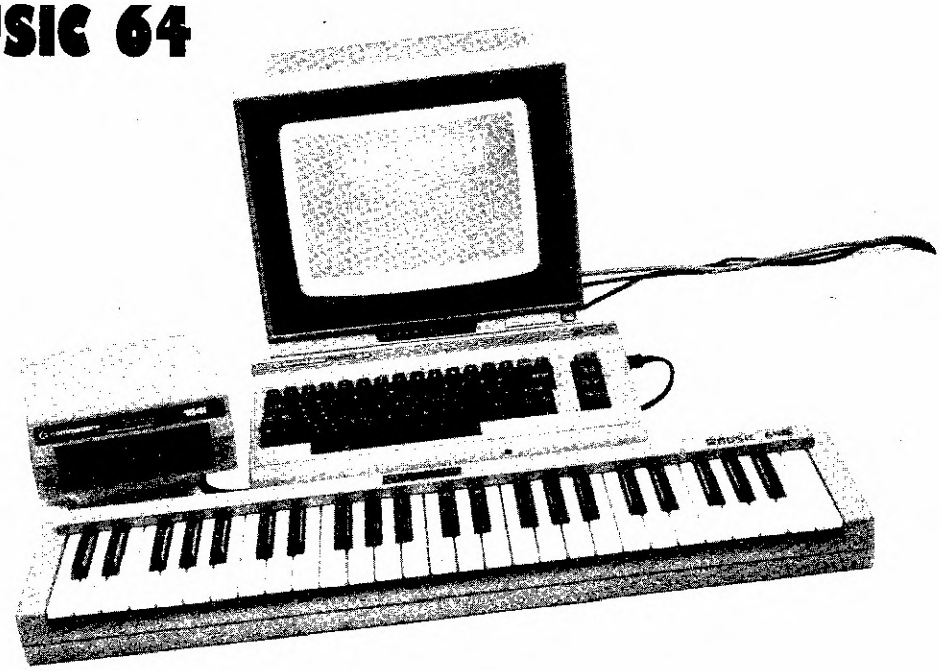
von Seite 61

```
4770 NEXT N
4780 RUN
4820 CLS
4830 PRINT AT 9,0;"*************
*******************"
4840 LET A$="II# BIS ZUM NAECHST
EN MAL, CIAO  #"
4860 GOSUB 5000
4850 PRINT AT 13,0;"************
*********************"
4870 GOTO 9999
5030 LET Z=VAL A$(1 TO 2)
5035 PRINT AT Z,0;"■";
5040 FOR N=3 TO LEN A$-1
5050 PRINT AT Z,N-3;A$(N);"■"
5060 NEXT N
5070 PRINT AT Z,N-3;A$(N)
5080 RETURN
```

WERSIBOARD MUSIC 64

für Commodore C 64/SX 64

Wersiboard Music 64, ein Keyboard mit Interface und kompletter Software (Diskette und Handbuch) für kreative Musik und aktives Musizieren.
Sie spielen auf einer echten Klavierstruktur fast alle Instrumente eines größeren Orchesters. Durch die zahlreichen Gestaltungsmöglichkeiten können beliebig viele Klangfarben erzeugt werden, bis hin zu den eindruckvollsten Synthesizer-Stimmen.
Mit dem neuen Software-Angebot lassen sich über das Wersiboard 64 bis zu 640 selbstentwickelte Klangfarben in zwei unterschiedlichen Bereichen speichern und über ein Inhaltsverzeichnis jederzeit abrufen. Damit gehört der Commodore 64 mit zu den universellsten Home-Computern in Verbindung mit Musikspeichern.



```
27880 :  40  17 254   2  40  20
27886 : 254   3  40  21 125 198
27892 :   5 111 254  33 200  24
27898 : 234 123 254   1  40   7
27904 :  24 240 123 254   3  40
27910 : 235   0  44  44  44  78
27916 :  45  70  22  71  62 128
27922 : 205 177 109  45 126 254
27928 :   1  40  24  12  62  32
27934 : 185  32  29  13  54   1

Summe: 5228

27940 :   4   4   4  62  14 184
27946 :  32  18  54   2  45  54
27952 :   0  24 191  13  62 255
27958 : 185  32   5  12  54   0
27964 :  24 230  44 112  44 113
27970 :  45  45  45  62  68 134
27976 :  87  62  43 134 205 177
27982 : 109  24 161  33  69  91
27988 : 126 254   1  40   9  35
27994 :  35  35 125 254  78  32

Summe: 4495

28000 : 243 201 229  35  70  35
28006 :  78  33   9  91 126 254
28012 :   2  32  11 125 198   5
28018 : 111 254  34  32 243 225
28024 :  24 223  35 126 184  40
28030 :   3  43  24 237  35 126
28036 : 185  40   4  43  43  24
28042 : 228  53  43  43  54   2
28048 :  43  94  54  91  42   2
28054 :  91   1  10   0   9  29

Summe: 5004

28060 :  32 252  34   2  91  58
28066 :   5  91 254   0  40   4
28072 :  61  50   5  91 205 239
28078 : 109  24 168 229 213 245
28084 : 197  33 141  92 114  62
28090 :   2 205   1  22  62  22
28096 : 215 193 197 120 215 121
28102 : 215 193 241 245 215  33
28108 : 141  92  54  71 241 209
28114 : 225 201 205  60  32  42

Summe: 7231

28120 :   0  91 237  75   2  91
28126 :  84 120 186  40   6 216
28132 : 237  67   0  91 201  85
28138 : 121 186 216  24 245  33
28144 :   0   2  17   1   0   6
28150 :  40 205 204 100  43  16
28156 : 250 209 225 213  54   0
28162 : 201   0   0   0   0   0
28168 :   0   0   0   0   0   0
28174 :   0   0   0   0   0   0

Summe: 4440
```

TEXAS INSTRUMENTS

Coconut Joe ist wieder da, denn hier kommt:

MAYA – 2

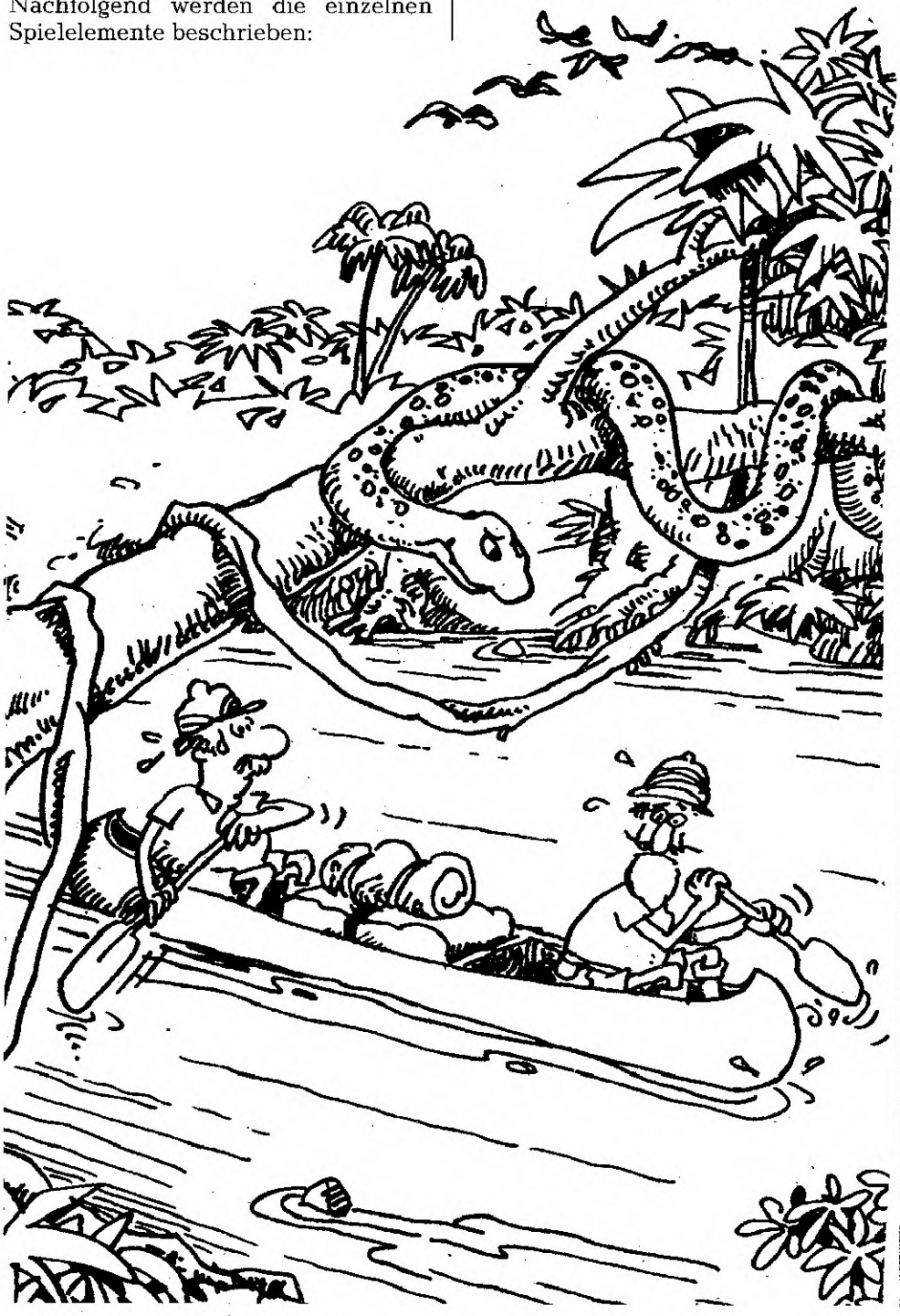
die Fortsetzung des im August 1984 veröffentlichten Programmes MAYA für den TI 99 / 4 A und den Extended Basic Modul.

Zu dem Programm MAYA bekam ich sehr viele Zuschriften und Anrufe begeisterter TI-User, die sich weitere Programme wie MAYA wünschten.
Aus diesem Grund entschloß ich mich, zu MAYA eine Fortsetzung, MAYA-2, zu schreiben.
Alle TI User, die dachten, mit 378 Bildern bei MAYA wäre die Grenze des Machbaren erreicht, sollten sich jetzt besser setzen, bevor sie weiter lesen.
Durch mehrere Ebenen bei MAYA-2 kann dieses Programm durch Kombination insgesamt 40768 verschiedene Bilder erzeugen. Dies hat zur Folge, daß Coconut Joe durch ein schier unendliches Labyrinth irren muß, um seine Schätze zu finden.
Wo MAYA endete, beginnt nun MAYA-2, nämlich in der Wüste: Coconut Joe ist weit in die Wüste vorgedrungen und hat eine sehr schatzträchtige Gegend entdeckt.
Auf dem Wüstenboden und in einem tiefen Katakombensystem wimmelt es nur so von Schätzen.
Coconut Joe muß jetzt so viele Schätze wie möglich sammeln. So weit so gut, aber das hätten wir ja bereits bei MAYA genauso. Der große Unterschied besteht jetzt darin, daß Coconut Joe auf einem Bildschirm gleich 3 Stockwerke vorfindet und durch Leitern den Bildschirm nach unten bzw. oben verlassen kann. Dadurch gelangt er in immer weitere Etagen der Kata-

komben mit immer mehr Schätzen.
Ferner lauern auf Coconut Joe neue Gefahren, wie z. B. Geister und Spinnen, die ihm den Garaus machen können.
Am gefährlichsten sind jedoch die Masken der verstorbenen Pharaonen, denn wenn Joe diese Masken berührt, stirbt er selbst im Sprung eines seiner drei Leben.
Coconut Joe kann Punkte sammeln, indem er Goldbarren, Silberbarren und Amulette aufnimmt.
Er bekommt für Silberbarren 50 Punkte, für Goldbarren 100 Punkte, für Amulette 200 Punkte.
Nachfolgend werden die einzelnen Spielelemente beschrieben:

Steuerung von Coconut Joe
Steuern Sie Coconut Joe mittels Joystick 1 nach rechts oder links sowie die Leitern hinauf oder hinunter.
Um eine Leiter zu betreten, ist der Joystick diagonal in die entsprechende Richtung zu drücken.
Zum Springen, wird der Aktionsknopf gedrückt.

Leitern
Über die Leitern ist es möglich das Stockwerk zu wechseln. Sind Sie jedoch vorsichtig, da die Leitern magisch aufgeladen sind und Coconut Joe anziehen. Es ist nicht leicht vor einer Leiter wieder herunter zu kommen. Dies bedarf einiger Übung.



Mauern
Coconut Joe kann nicht vor einer Mauer vorbeigehen, da er sonst stirbt. Er muß vor einer Mauer vorbeispringen. Sind Sie jedoch beim Springen vorsichtig, wenn sich eine Fallgrube direkt hinter der Mauer befindet, sonst springt Joe direkt in diese Fallgrube und stirbt eines seiner drei Leben.

Fallgruben
Wenn Joe über eine Fallgrube geht, stürzt er ab und stirbt. Joe muß über die Fallgrube springen.
Wenn eine Leiter in der Fallgrube steht, kann Joe dort die Leiter betreten. Dazu muß der Joystick jedoch diagonal gedrückt werden, denn sonst stürzt Joe ab.

Geister
Geister kommen nur in den Katakomben vor. Coconut Joe kann diese Geister überspringen.

Spinnen
Die Spinnen kommen ebenfalls nur in den Katakomben vor, und können auch übersprungen werden.

Kakteen und rollende Baumstämme
Kakteen und Baumstämme kommen nur an der Wüstenoberfläche vor und müssen übersprungen werden, da sonst Joe eines seiner drei Leben verliert.

Pharaonenmasken
Diese Masken müssen umgangen werden, da eine Berührung tödlich ist. Selbst wenn eine Maske übersprungen wird, stirbt Joe den Abenteurertod.

Silberbarren, Goldbarren und Amulette
Um einen solchen Schatz aufzunehmen, muß Joe diesen einfach nur berühren.
Nachdem die Elemente beschrieben sind, können Sie das Spiel mit „RUN" starten. Nach einiger Zeit erscheint das Titelbild auf dem Bildschirm und Amazing Grace wird gespielt. Ein Druck auf den Aktionsknopf von Joystick 1 startet das Spiel.
Nachdem Joe alle 3 Leben verloren hat, erscheint wieder das Titelbild. Ein neues Spiel wird durch einen Druck auf Aktionsknopf 1 gestartet.

Anmerkung
Einige Programmzeilen sind so lang, daß sie nicht auf einmal eingegeben werden können. Geben Sie die Zeile ein, bis Ihr TI piepst. Jetzt schicken Sie die Zeile mit (ENTER) ab und holen diese mit (REDO) wieder zurück. Danach können Sie den Rest der Zeile eingeben.
Diskettenbenutzer ohne Speichererweiterung müssen die Peripheriebox abgeschaltet lassen, da sonst der Speicherplatz nicht ausreicht. Das Programm belegt bis auf 780 Bytes den kompletten Speicher.
Während des Spieles darf die (ALPHA LOCK) Taste nicht gedrückt sein, da sonst der Joystick nicht richtig abgefragt wird.

```
100 ! ********************
110 ! *      MAYA - 2    *
120 ! *------------------*
130 ! * TI 99/4 A + XBASIC *
140 ! * Steuerung: Joyst.1 *
150 ! *------------------*
160 ! *    40768 Bilder  *
170 ! *------------------*
180 ! * (c) 1985  V.Becker *
190 ! * Steinbacher Str.10 *
200 ! *                  *
210 ! ********************
220 !
230 !
240 DATA 4,195,261,329,4,329,523,1,329,659,1,329,523,4,391,659,2,415,587
250 DATA 2,174,220,261,4,440,523,2,349,440,4,195,261,329,4,329,391,2,391,30000,4
,329,523,1,329,659,1,329,523
260 DATA 2,195,246,349,4,391,659,2,391,587,2,195,261,329,6,523,783,2,195,246,349
,4,493,783,2,493,659
270 DATA 4,195,261,329,5,523,783,1,523,659,4,391,523,2,391,30000,2,174,220,261,4
,349,440,1,349,523,1,349,440
280 DATA 4,195,261,329,4,329,391,2,30000,391,4,329,523,1,329,659,1,329,523
290 DATA 1,195,246,293,3,391,659,1,195,246,349,1,391,659,2,391,587,1,195,261,329
,3,329,523,1,174,220,261
300 DATA 1,329,523,2,349,523,2,195,261,329,4,329,523,2,30000,391,0,0
310 OPTION BASE 1
320 DIM ST(3),B(3)
330 CALL CLEAR :: CALL SCREEN(2):: CALL GRAPHIC :: CALL BILD :: CALL PYRAMIDE(I,
ST()):: CALL MAGNIFY(3)
340 RANDOMIZE :: VG$=RPT$("0",143):: PKT=0
350 CALL PUNKTE(PKT):: CALL HCHAR(1,26,62,3):: LB=3 :: CALL TITEL(B(),PS,VG$,I,S
T()):: GOSUB 360 :: GOTO 420
360 ON INT(RND*5+1.5)GOSUB 1150,1160,1170,1180,1190,1200
370 ON INT(RND*5+1.5)GOSUB 1090,1100,1110,1120,1130,1140
380 ON INT(RND*5+1.5)GOSUB 1030,1040,1050,1060,1070,1080
390 FOR I=2 TO 3 :: ON INT(RND*6+1.5)GOSUB 1230,1220,1210,1240,1250,1260,1270
400 NEXT I
410 RETURN
420 CALL PYRAMIDE(I,ST()):: EBENE=1 :: STOCK=1 :: CALL SPRITE(#1,68,13,(7*STOCK)
*8-8,232,0,0)
430 CALL JOYST(1,X,Y):: IF Y=4 THEN 770
440 IF X=0 THEN 460
450 CALL SOUND(-5,523,0):: IF X<0 THEN CALL PATTERN(#1,68)ELSE CALL PATTERN(#1,8
4)
460 CALL MOTION(#1,0,X*2)
470 CALL POSITION(#1,X,Y):: IF Y<16 OR Y>232 THEN 910
480 CALL GCHAR((X+13)/8,(Y+9)/8,BV):: IF SEG$(VG$,BV,1)="1" THEN 580
490 CALL GCHAR((X-17)/8,(Y+9)/8,BV):: IF BV>119 THEN IF SEG$(VG$,BV,1)="1" THEN
990
500 CALL KEY(1,T,S):: IF T=18 THEN 870
510 CALL JOYST(1,X,Y):: IF Y=4 THEN 720
520 IF X=0 THEN 540
530 CALL SOUND(-5,261,0):: IF X<0 THEN CALL PATTERN(#1,76)ELSE CALL PATTERN(#1,9
2)
540 CALL MOTION(#1,0,X*2)
550 CALL PEEK(-31877,KOL):: IF (KOL AND 32)=32 THEN 950
560 CALL KEY(1,T,S):: IF T=18 THEN 870
570 GOTO 430
580 CALL GCHAR(INT((X+5)/8+3),(Y+9)/8,BV):: IF SEG$(VG$,BV,1)="0" THEN 950
590 CALL JOYST(1,X,Y):: IF Y<>-4 THEN 990
600 CALL PATTERN(#1,132):: CALL MOTION(#1,0,0):: CALL POSITION(#1,X,Y):: CALL LO
CATE(#1,X+24,121):: Y2=X :: X2=Y
610 CALL JOYST(1,X,Y):: CALL POSITION(#1,Y1,X1):: CALL GCHAR((Y1+9)/8,(X1+9)/8,B
```

```
V):: IF BV>71 THEN CALL LOCATE(#1,Y3,X3):: CALL MOTION(#1,0,0):: GOTO 800
620 CALL MOTION(#1,-Y*1.5,0):: CALL POSITION(#1,Y3,X3)
630 CALL PEEK(-31877,KOL):: IF (KOL AND 32)=32 THEN 990
640 CALL POSITION(#1,Y2,X2):: IF Y2>184 OR Y2<17 THEN 830
650 CALL GCHAR((Y2+9)/8,(X2+9)/8,BV):: IF BV>71 THEN 670
660 GOTO 610
670 CALL JOYST(1,Y,X):: IF Y=0 THEN 610
680 CALL POSITION(#1,Y1,X1):: STOCK=INT((Y1-24)/56)+1
690 CALL GCHAR((Y1+9)/8,(X1+9)/8,BV):: IF BV>71 AND BV<96 AND SEG$(VG$,BV,1)="0"
 THEN 610
700 CALL MOTION(#1,0,0):: IF Y=-4 THEN CALL SPRITE(#1,68,13,(7*STOCK)*8-8,X1-12)
ELSE CALL SPRITE(#1,84,13,(7*STOCK)*8-8,X1+12)
710 GOTO 430
720 CALL POSITION(#1,X1,Y1)
730 CALL GCHAR((X1-17)/8,(Y1+9)/8,BV):: IF BV<70 THEN IF SEG$(VG$,BV,1)="1" THEN
 750
740 GOTO 520
750 CALL PATTERN(#1,132):: CALL MOTION(#1,0,0):: CALL LOCATE(#1,X1-17,121)
760 GOTO 610
770 CALL POSITION(#1,X1,Y1)
780 CALL GCHAR((X1-17)/8,(Y1+9)/8,BV):: IF BV<70 THEN IF SEG$(VG$,BV,1)="1" THEN
 750
790 GOTO 440
800 IF Y=4 AND BV<96 THEN 630
810 IF Y=-4 AND BV>95 THEN 630
820 GOTO 620
830 CALL DELSPRITE(ALL):: CALL MOTION(#1,0,0):: IF Y2>184 THEN EBENE=EBENE+1 ELS
E EBENE=EBENE-1
840 IF EBENE=1 THEN CALL PYRAMIDE(I,ST())ELSE CALL KATAKOMBE(I,ST())
850 GOSUB 380 :: IF Y2>184 THEN CALL SPRITE(#1,132,13,17,X2)ELSE CALL SPRITE(#1,
132,13,184,X2)
860 GOTO 610
870 CALL POSITION(#1,X,Y):: CALL LOCATE(#1,X-10,Y)
880 FOR I=110 TO 160 STEP 10 :: CALL SOUND(-100,I,0,I+2,0,I+4,0):: CALL PEEK(-31
877,KOL):: IF ((KOL AND 32)=32)AND(ST(STOCK)=5)THEN 990
890 NEXT I
900 CALL POSITION(#1,X,Y):: CALL LOCATE(#1,X+10,Y):: GOTO 430
910 CALL DELSPRITE(ALL):: IF EBENE=1 THEN CALL PYRAMIDE(I,ST())ELSE CALL KATAKOM
BE(I,ST())
920 CALL MOTION(#1,0,0)
930 GOSUB 360 :: IF Y<16 THEN CALL SPRITE(#1,68,13,X,232)ELSE CALL SPRITE(#1,84,
13,X,16)
940 GOTO 430
950 ON ST(STOCK)+1 GOTO 430,960,970,980,990,990
960 PKT=PKT+50 :: CALL DELSPRITE(#STOCK+1):: CALL PUNKTE(PKT):: CALL GONG(1):: G
OTO 430
970 PKT=PKT+100 :: CALL DELSPRITE(#STOCK+1):: CALL PUNKTE(PKT):: CALL GONG(2)::
GOTO 430
980 PKT=PKT+200 :: CALL DELSPRITE(#STOCK+1):: CALL PUNKTE(PKT):: CALL GONG(3)::
GOTO 430
990 CALL MOTION(#1,0,0):: CALL PATTERN(#1,132):: CALL MOTION(#1,6,0):: CALL LEBE
N(LB):: CALL KAMERAD :: IF LB=0 THEN 1010
1000 CALL SPRITE(#1,68,13,(7*STOCK)*8-8,232,0,0):: STOCK=1 :: GOTO 430
1010 GOSUB 1190 :: GOSUB 1200 :: LB=3 :: PKT=0 :: CALL TITEL(B(),PS,VG$,I,ST())
1020 CALL HCHAR(1,1,58,32):: CALL HCHAR(1,26,62,3):: CALL PUNKTE(PKT):: GOTO 420
1030 CALL COLOR(12,7,2):: PS=120 :: GOSUB 1300 :: RETURN
1040 CALL COLOR(13,7,2):: PS=128 :: GOSUB 1300 :: RETURN
1050 CALL COLOR(14,7,2):: PS=136 :: GOSUB 1300 :: RETURN
1060 CALL COLOR(12,2,2):: PS=120 :: GOSUB 1310 :: RETURN
1070 CALL COLOR(13,2,2):: PS=128 :: GOSUB 1310 :: RETURN
1080 CALL COLOR(14,2,2):: PS=136 :: GOSUB 1310 :: RETURN
1090 CALL COLOR(6,7,2,11,11,2):: PS=72 :: GOSUB 1300 :: PS=112 :: GOSUB 1300 ::
RETURN
```

```
1100 CALL COLOR(7,7,2,10,11,2):: PS=80 :: GOSUB 1300 :: PS=104 :: GOSUB 1300 ::
RETURN
1110 CALL COLOR(8,7,2,9,11,2):: PS=88 :: GOSUB 1300 :: PS=96 :: GOSUB 1300 :: RE
TURN
1120 CALL COLOR(6,7,7,11,11,11):: PS=72 :: GOSUB 1310 :: PS=112 :: GOSUB 1310 ::
 RETURN
1130 CALL COLOR(7,7,7,10,11,11):: PS=80 :: GOSUB 1310 :: PS=104 :: GOSUB 1310 ::
 RETURN
1140 CALL COLOR(8,7,7,9,11,11):: PS=88 :: GOSUB 1310 :: PS=96 :: GOSUB 1310 :: R
ETURN
1150 CALL COLOR(1,8,2):: PS=32 :: GOSUB 1300 :: PS=33 :: GOSUB 1300 :: RETURN
1160 CALL COLOR(2,8,2):: PS=40 :: GOSUB 1300 :: PS=41 :: GOSUB 1300 :: RETURN
1170 CALL COLOR(5,8,2):: PS=64 :: GOSUB 1300 :: PS=65 :: GOSUB 1300 :: RETURN
1180 CALL COLOR(1,2,2):: PS=32 :: GOSUB 1310 :: PS=33 :: GOSUB 1310 :: RETURN
1190 CALL COLOR(2,2,2):: PS=40 :: GOSUB 1310 :: PS=41 :: GOSUB 1310 :: RETURN
1200 CALL COLOR(5,2,2):: PS=64 :: GOSUB 1310 :: PS=65 :: GOSUB 1310 :: RETURN
1210 CALL SPRITE(#I+1,116,16,(7*I)*8-8,24,0,12):: ST(I)=4 :: RETURN
1220 CALL SPRITE(#I+1,124,14,(7*I)*8-8,224,0,-12):: ST(I)=4 :: RETURN
1230 CALL SPRITE(#I+1,140,5,(7*I)*8-8,84,0,0):: ST(I)=5 :: RETURN
1240 CALL SPRITE(#I+1,36,7,(7*I)*8-8,84,0,0):: ST(I)=2 :: RETURN
1250 CALL SPRITE(#I+1,36,16,(7*I)*8-8,156,0,0):: ST(I)=1 :: RETURN
1260 CALL SPRITE(#I+1,44,2,(7*I)*8-8,156,0,0):: ST(I)=3 :: RETURN
1270 CALL DELSPRITE(#I+1):: ST(I)=0 :: RETURN
1280 CALL SPRITE(#I+1,108,2,(7*I)*8-8,84,0,0):: ST(I)=4 :: RETURN
1290 CALL SPRITE(#I+1,100,7,(7*I)*8-8,24,0,12):: ST(I)=4 :: RETURN
1300 VG$=SEG$(VG$,1,PS-1)&"1"&SEG$(VG$,PS+1,143):: RETURN
1310 VG$=SEG$(VG$,1,PS-1)&"0"&SEG$(VG$,PS+1,143):: RETURN
1320 SUB AMAZINGGRACE(B(),KY)
1330 CALL SOUND(800,391,0)
1340 RESTORE
1350 AZ,STV=0
1360 READ K :: IF K=0 THEN 1340
1370 READ B(1),B(2),B(3)
1380 FOR I=1 TO K
1390 FOR J=1 TO 3
1400 STV=STV+1 :: IF STV>AZ THEN READ AZ,M1,M2 :: STV=1
1410 CALL SOUND(400,M1,1,M2,0,B(J),5):: CALL KEY(1,T,S):: IF S<>0 THEN SUBEXIT
1420 NEXT J :: NEXT I
1430 GOTO 1360
1440 SUBEND
1450 SUB PUNKTE(PKT)
1460 P$=STR$(PKT):: DISPLAY AT(1,8)SIZE(LEN(P$)):P$ :: SUBEND
1470 SUB GONG(VAR)
1480 CALL MOTION(#1,0,0)
1490 FOR I=0 TO 9 :: CALL SOUND(-250,391,I):: NEXT I
1500 IF VAR=1 THEN SUBEXIT
1510 FOR I=0 TO 9 :: CALL SOUND(-250,329,I,391,I+11):: NEXT I
1520 FOR I=0 TO 9 :: CALL SOUND(-250,261,I,329,I+11,391,I+21):: NEXT I
1530 IF VAR=2 THEN SUBEXIT
1540 FOR I=0 TO 9 :: CALL SOUND(-250,391,I,261,I+11,329,I+21):: NEXT I
1550 FOR I=0 TO 9 :: CALL SOUND(-250,261,I,391,I+11,329,I+21):: NEXT I
1560 SUBEND
1570 SUB TITEL(B(),PS,VG$,I,ST())
1580 CALL PYRAMIDE(1,ST()):: CALL DELSPRITE(ALL):: CALL MAGNIFY(4)
1590 FOR I=1 TO 7 :: ON I GOSUB 1030,1040,1050,1090,1100,1110,1150
1600 NEXT I
1610 DATA ,0103060C183060C,0103070F1B3363C3,83,0103070F1B3363C3,83030303030303
,0F0F1830303030303,303F3F303030303,E1E1311919191818
1620 DATA 18F8F81818181818,8C8C8C8C8C8CD8D8,707070707070707,80C0E0F0D8CCC6C3,C1C
0FFFFC0C0C0C,,80C0E0F0180C0603
1630 DATA 0003070C18303,0000000001030 70F,00C0E030180C0C0C,183060C08000FCFC
1640 RESTORE 1610
1650 FOR I=108 TO 140 STEP 8 :: FOR K=0 TO 3 :: READ C$ :: CALL CHAR(I+K,C$):: N
```

```
EXT K :: NEXT I
1660 P=64 :: FOR I=1 TO 4 :: CALL SPRITE(#I,8*I+100,16,65,P):: P=P+32 :: NEXT I
1670 CALL SPRITE(#5,140,16,121,112)
1680 CALL AMAZINGGRACE(B(),1)
1690 CALL DELSPRITE(ALL):: CALL MAGNIFY(3)
1700 RESTORE 2140
1710 FOR I=102 TO 143 :: READ C$ :: CALL CHAR(I,C$):: NEXT I
1720 SUBEND
1730 SUB LEBEN(LB)
1740 CALL HCHAR(1,25+LB,63):: LB=LB-1 :: SUBEND
1750 SUB BILD
1760 CALL VCHAR(1,1,30,768)
1770 DATA 2,2,2,2,2,2,11,5,11,5,2,2,7,7,7,7,7,7,11,11,11,11,11,11,2,2,2,2,2,2
1780 RESTORE 1770 :: FOR I=0 TO 14 :: READ V,H :: CALL COLOR(I,V,H):: NEXT I
1790 CALL HCHAR(1,1,58,32):: FOR I=6 TO 20 STEP 7 :: CALL HCHAR(I,1,59,96):: NEX
T I
1800 FOR I=9 TO 23 STEP 7 :: CALL HCHAR(I,1,73,32):: NEXT I
1810 DATA 120,128,136,120,104,80,96,88,112,72,112,72,104,80,112,72,96,88,112,72,
96,88,40,64
1820 FOR I=10 TO 17 STEP 7 :: FOR K=5 TO 28 STEP 23 :: READ M :: CALL VCHAR(I,K,
M,3):: NEXT K :: NEXT I
1830 FOR I=7 TO 21 STEP 7 :: FOR K=7 TO 25 STEP 9 :: READ C,C2 :: CALL VCHAR(I,K
,C,2):: CALL VCHAR(I,K+1,C,2):: CALL HCHAR(I+2,K,C2,2):: NEXT K :: NEXT I
1840 FOR I=10 TO 17 STEP 7 :: READ L :: CALL VCHAR(I,16,L,3):: CALL VCHAR(I,17,L
+1,3):: NEXT I
1850 DISPLAY AT(24,14)SIZE(2):" !"
1860 SUBEND
1870 SUB PYRAMIDE(I,ST())
1880 DISPLAY AT(2,1):":::<=:::::::::::::<=::::::::::<;;=:::::<=::::<;;=::::::::
::::::::<;;=::<;;;;=::<=:::::::::::::::::::::::::::<;;=:"
1890 FOR I=1 TO 2 :: CALL VCHAR(2,I,58,4):: CALL VCHAR(2,I+30,58,4):: NEXT I
1900 I=1 :: ON INT(RND*5+1.5)GOSUB 1290,1270,1280,1240,1250,1260
1910 SUBEND
1920 SUB KATAKOMBE(I,ST())
1930 CALL HCHAR(3,1,30,96):: CALL HCHAR(2,1,73,32):: CALL VCHAR(3,5,128,3):: CAL
L VCHAR(3,28,136,3):: CALL VCHAR(3,16,32,3)
1940 CALL VCHAR(3,17,33,3):: DISPLAY AT(2,5)SIZE(20):"PPIIIIIIIHHIIIIIIIPP"
1950 I=1 :: ON INT(RND*6+1.5)GOSUB 1230,1220,1210,1240,1250,1260,1270
1960 SUBEND
1970 SUB KAMERAD
1980 DATA 500,130,500,174,250,220,25,30000,100,220,25,30000,500,220,500,195,1000
,174,500,130
1990 RESTORE 1980
2000 CALL DELSPRITE(#1)
2010 FOR I=1 TO 10 :: READ L,T :: CALL SOUND(L,T,0):: NEXT I
2020 SUBEND
2030 SUB GRAPHIC
2040 DATA 1F1F18181F1F1818,F8F81818F8F81818,,,0000000103070F1D,3E7DFFD7ABFE,007F
FFD7AF96BC38
2050 DATA B060C08,1F1F18181F1F1818,F8F81818F8F81818,,,030C30404040402,1806010001
0205
2060 DATA E018060101010102,0C30C080C0A0508,0038444444444438,0010301010101038,003
844040810207C,0038440418044438,00081828487C0808
2070 DATA 007C407804044438,0018204078444438,007C04081020202,0038444438444438,003
844443C04083,,FFFFFFFFFFFFFFFF
2080 DATA 0103070F1F3F7FFF,80C0E0F0F8FCFEFF,0C1C0C142C1C1C14,18187E7E18181824,1F
1F18181F1F1818,F8F81818F8F81818,
2090 DATA ,0001000000C0201,0103030301020C,E0E0A0E0202060E,E0E0E0E0404040C,FFFF,FF
FFFFFFFFFFFFFF
2100 DATA F0F0F0F0F8F8F8F,,0001000000000708,0103030301010103,E0E0A0E02020E0E,E0E
0E0E04020101,FFFF
2110 DATA ,,,0707050704040607,0707070702020203,0080000030408,80C0C0C080403
2120 DATA FFFF,FFFF,,,0707050704040707,0707070702040808,0080000000000E01
```

Fortsetzung S. 78

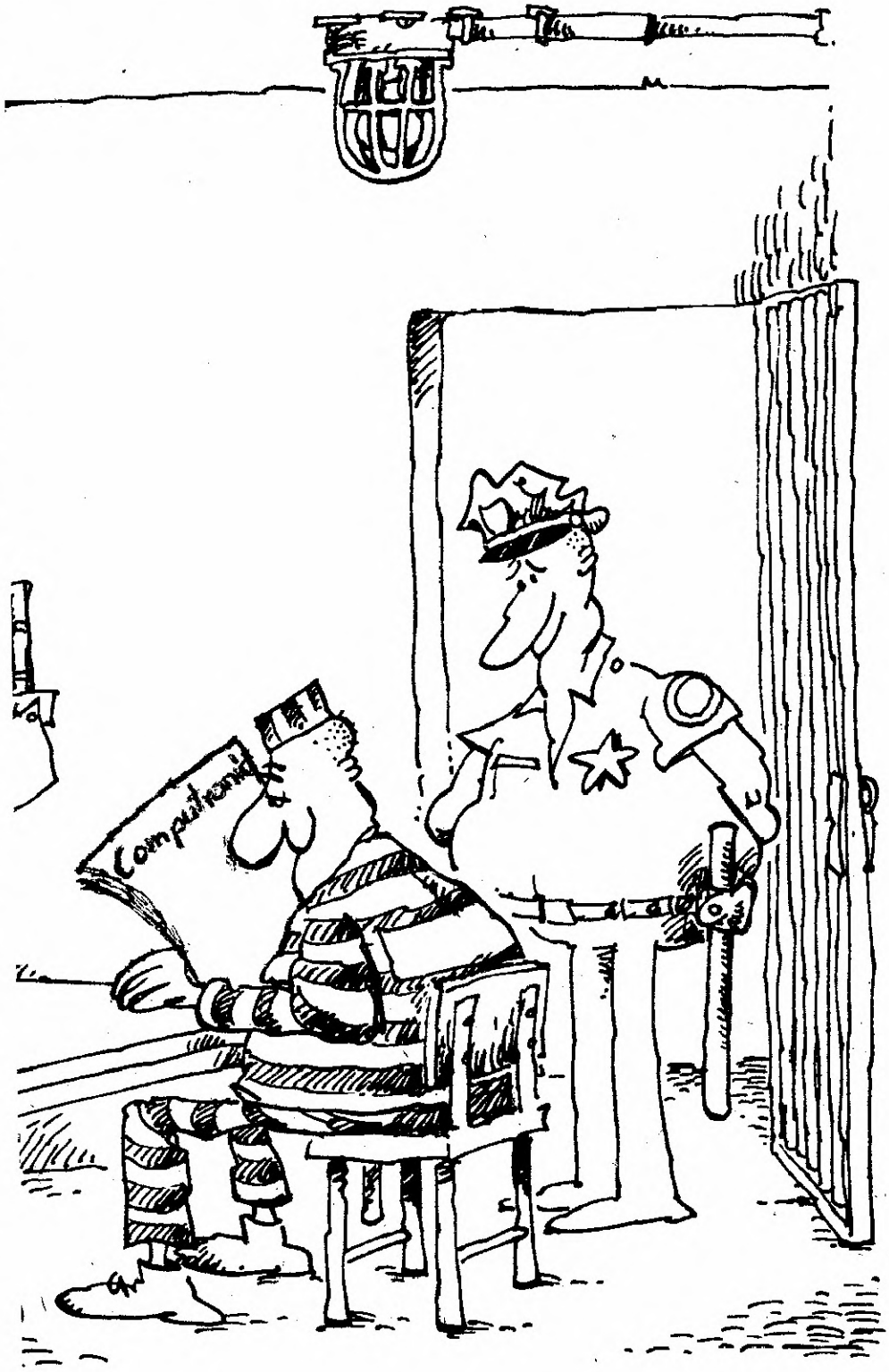
Jagdszene Chicago

Der gefürchtetste Ausbrecher Chicagos, Dale Dieb, ist aus dem Gefängnis ausgebrochen. Sie, als Kommissar Capture, haben nun die Aufgabe, diesen Bösewicht zu fangen.

Nicht selten kommt es vor, daß in Chicago ein Bösewicht das Weite sucht. Diesmal ist es aber der berüchtigtste Obergauner von ganz Chicago. Bringen Sie ihn also wieder in die Arme der Gerechtigkeit zurück, bevor er Banken, Schmuckgeschäfte und Goldhändler überfällt.

Programmablauf

Nach dem Starten von Teil 1 erklingt die Titelmelodie, anschließend wird man aufgefordert, den 2. Teil zu laden. Ist dies geschehen, wird man gefragt, ob man mit dem Joystick spielen möchte oder ob man über die Tastatur spielen will. Man bewegt dazu das Kästchen mit den Pfeiltasten in die entsprechende Richtung und drückt Taste Q zum Stoppen des Kästchens an entsprechender Stelle. Bei Spielbeginn ist das grüne Männchen der Dieb. Ist es dem blauen Männchen, dem Polizisten, gelungen, den Dieb zu fangen, wechseln die Rollen. Die Joysticks oder die Tastaturseiten brauchen nicht gewechselt zu werden. Der Polizist hat natürlich einen Vorteil. Er kann durch jene Wände hindurchschlüpfen, die hellblau gefärbt sind.
Ist das Spiel beendet, werden die Punktergebnisse, der Gewinner und der High-Score aufgelistet. Möchten Sie ihren Partner noch einmal durch Chicago jagen, drücken Sie „J", ansonsten „N".
Um einen einigermaßen guten High-Score zu erzielen, brauchen Sie auch ein wenig Glück, da der Computer die Punkte zum Teil zufällig wählt.
Tippen Sie zuerst Teil 1 ab, und testen Sie ihn. Dazu sollten Sie aber den Befehl „RUN CS1" in Zeile 370 vorerst weglassen. Läuft Teil 1 fehlerlos, so können Sie die Zeile 370 um „RUN CS1" ergänzen. Jetzt können Sie den 1. Teil abspeichern. Nun tippen Sie den 2. Teil ab. Läuft auch er fehlerfrei, können Sie ihn abspeichern. Erschrecken Sie aber nicht, wenn Sie Teil 2 testen. Auf dem Bildschirm entsteht ein großes Wirrwarr, da Teil 1 alle notwendigen Definitionen enthält.



```
1 ! ************************
2 ! *                      *
3 ! *  JAGDSZENE CHICAGO   *
4 ! *                      *
5 ! * (C)1985 BY CHRISTOPH *
6 ! *     CRONIMUND        *
7 ! *                      *
8 ! *  CH-8304 WALLISELLEN *
9 ! *                      *
10 !*                      *
11 !************************
100 CALL CLEAR :: CALL SCREEN(2):: RANDOMIZE :: FA=INT(RND*14)+1
110 FOR A=5 TO 8 :: CALL COLOR(A,FA+2,2):: NEXT A
120 RESTORE 380 :: FOR ZE=1 TO 26 :: READ ZE$ :: CALL CHAR(ZE+64,ZE$):: NEXT ZE
130 CALL CHAR(40,"FFFF00000000FFFF")
140 CALL CHAR(41,"C3C3C3C3C3C3C3C3")
150 CALL CHAR(42,"003C7E66C3C3C3C3",43,"0F3F706060703F0F",44,"C3C3C3C3667E3C",45
,"F0FC0E06060EFCF")
160 CALL CHAR(46,"0F3F7060C0C0C1C3",47,"C3C1C0C060703F0F")
170 CALL CHAR(91,"C3C3030303C3C3",92,"FFFF00000000C3C3")
180 CALL CHAR(136,"0F3F7060C0C0C0C0C0C0C0C060703F0FFFFF000000000000000000000000F
FFF")
190 CALL CHAR(140,"FFFF000000000000000000000000FFFFF0FC0E06030303030303030360EF
CF")
200 CALL CHAR(95,"183C66C3C3663C18")
210 A$="0F3F7060C0C0C0C" :: B$="C0C0C0C060703F0F" :: C$="030303030060EFCF" :: D$=
"F0FC0E0603030303"
220 E$="FFFF" :: F$="C0C0C0C0C0C0C0C" :: G$="0000000000000FFFF" :: H$="0303030303
030303"
230 CALL CHAR(96,A$,97,B$,98,C$,99,D$,100,E$,101,F$,102,G$,103,H$)
240 CALL CHAR(104,A$,105,B$,106,C$,107,D$,108,E$,109,"FC020201010202FC",110,G$,1
11,H$)
250 CALL CHAR(112,F$,113,"3F4040808040403F",123,"00183C7E7E3C18")
260 CALL CHAR(120,"18187E18187E4242")
270 CALL CHAR(121,"18181E18183C2412",122,"18187818183C2448")
280 CALL CHAR(124,"287CAAA87C2AAA7C",125,"00182C5E5E3C18",126,"0000003C7E7E",127
,"00FE92AA92FE")
290 CALL CHAR(88,"3C42999919199423C")
300 CALL CHAR(81,"003C7EFFFFFF7E3C"):: DISPLAY AT(1,5):"XCRONSOFT PRESENTS"
310 DISPLAY AT(4,5):"QQQQQQQQQQQQQQQQQQQQ" :: DISPLAY AT(5,4):"QQQQQQQQQQQQQQQQQQ
QQQ"
320 DISPLAY AT(6,4):"Q                    Q" :: DISPLAY AT(7,4)BEEP:"Q JAGDSZENE
CHICAGO Q" :: DISPLAY AT(8,4):"Q                    Q"
330 DISPLAY AT(9,4):"QQQQQQQQQQQQQQQQQQQQ" :: DISPLAY AT(10,5):"QQQQQQQQQQQQQQQQ
QQQQ"
340 DISPLAY AT(13,13):"BY" :: DISPLAY AT(15,5):"CHRISTOPH CRONIMUND"
350 DISPLAY AT(24,3):"PROGRAMMSTART MIT ENTER"
360 CALL MELODIE
370 CALL CLEAR :: RUN "CS1"
380 DATA 00183C667E666666,007C66667C66667C,003C66606060663C,00786C6666666C78
390 DATA 007E60607860607E,007E60607860606 0,003C66606C66663C,006666667E666666
400 DATA 003C18181818183C,0006060606066663C,00666C6C786C6C66,006060606060607E
410 DATA 00667E7E66666666,006676766E6E6666,003C66666666663C,007C66667C606060
420 DATA 003C666666766E3C,007C66667C786C66,003C66603C06663C,007E181818181818
430 DATA 00666666666663C,00666666666663C18,00666666667E7E24,0066663C183C6666
440 DATA 006666663C181818,007E060C1830607E
450 SUB MELODIE
460 W=0
470 RESTORE 570 :: FOR ME=1 TO 14 :: READ TL,T1,LA :: CALL SOUND(TL,T1,LA)
480 CALL KEY(3,CO,ST):: IF CO>-1 THEN SUBEXIT
490 NEXT ME :: W=W+1 :: IF W=2 THEN 500 ELSE 470
500 W=0
510 RESTORE 590 :: FOR ME=1 TO 10 :: READ TL,T1,LA :: CALL SOUND(TL,T1,LA)
```

```
520 CALL KEY(3,CO,ST):: IF CO>-1 THEN SUBEXIT
530 NEXT ME :: W=W+1 :: IF W=2 THEN 540 ELSE 510
540 RESTORE 610 :: FOR ME=1 TO 6 :: READ TL,T1,LA :: CALL SOUND(TL,T1,LA)
550 CALL KEY(3,CO,ST):: IF CO>-1 THEN SUBEXIT
560 NEXT ME :: FOR V=1 TO 600 :: NEXT V :: GOTO 460
570 DATA 330,392,0,330,523,0,330,523,0,330,587,0,330,587,0,440,659,0,150,698,0
580 DATA 300,784,0,330,698,0,330,659,0,330,659,0,330,587,0,330,587,0,440,523,0
590 DATA 150,784,0,150,698,0,330,659,0,330,659,0,330,659,0,150,698,0,150,659,0
600 DATA 330,587,0,330,587,0,330,587,0
610 DATA 330,784,0,330,659,0,330,523,0,330,587,0,330,587,0,660,523,0
620 SUBEND
```

```
1 !*************************
2 !*      LISTING 2        *
3 !*                       *
4 !*   JAGDSZENE CHICAGO   *
5 !*         BY            *
6 !*  CHRISTOPH CRONIMUND  *
7 !*  CH-8304 WALLISELLEN  *
8 !*************************
110 DIM WA(2),PU(2),ZA$(10),ZE$(26):: WE=1 :: PU(1)=0 :: PU(2)=0 :: PL=0 :: CALL
 CLEAR :: CALL SCREEN(2)
120 RANDOMIZE :: CALL COLOR(8,2,2)
130 RESTORE 1470 :: FOR ZA=1 TO 10 :: READ ZA$(ZA):: NEXT ZA
140 CALL CHARPAT(90,MUST$):: CALL CHAR(82,MUST$)
150 RESTORE 1400 :: FOR ZE=1 TO 26 :: READ ZE$(ZE):: NEXT ZE
160 FOR ZE=1 TO 26 :: CALL CHAR(ZE+64,ZE$(ZE)):: NEXT ZE
170 CALL MAGNIFY(3)
180 FOR C=5 TO 8 :: CALL COLOR(C,16,2):: NEXT C :: DISPLAY AT(7,1):"SPIELEN SIE
MIT DEM JOYSTICK"
190 CALL HCHAR(7,1,123):: CALL HCHAR(7,32,123)
200 DISPLAY AT(9,4):"JA" :: DISPLAY AT(9,22):"NEIN"
210 CALL SPRITE(#1,136,8,62,112):: CALL SPRITE(#2,140,8,62,126)
220 CALL KEY(1,CO,ST):: IF CO=2 THEN X=-4 ELSE IF CO=3 THEN X=4 ELSE IF CO=18 TH
EN 250 ELSE 220
230 CALL MOTION(#1,0,X*2):: CALL MOTION(#2,0,X*2)
240 GOTO 220
250 CALL MOTION(#1,0,0):: CALL MOTION(#2,0,0):: CALL POSITION(#1,Y1,X1,#2,Y2,X2)
:: IF X1<128 THEN JO=1 ELSE JO=0
260 CALL DELSPRITE(#1,#2):: CALL MAGNIFY(1)
270 DISPLAY AT(7,1):"                                  " :: CALL COLOR(8,2,2)
280 DISPLAY AT(9,4):"                                  "
290 FOR ZA=1 TO 10 :: CALL CHAR(ZA+47,ZA$(ZA)):: NEXT ZA :: CALL CHAR(58,ZE$(21)
,59,ZE$(16))
300 RESTORE 1230 :: FOR A=1 TO 24 :: READ S,Z,L :: CALL HCHAR(S,Z,40,L):: NEXT A
310 RESTORE 1260 :: FOR B=1 TO 22 :: READ S,Z,L :: CALL VCHAR(S,Z,41,L):: NEXT B
320 RESTORE 1290 :: FOR D=1 TO 35 :: READ S,Z,C :: CALL HCHAR(S,Z,C):: NEXT D
330 CALL CHAR(88,"C383030360EFCF",89,"F0FC0E06030383C3")
340 RESTORE 1340 :: FOR E=1 TO 26 :: READ S,Z,C :: CALL HCHAR(S,Z,C):: NEXT E
350 CALL CHAR(90,"C3C3C0C0C0C0C3C3")
360 CALL HCHAR(10,2,90):: CALL HCHAR(10,32,91):: CALL HCHAR(10,6,92):: CALL HCHA
R(10,28,92):: CALL HCHAR(18,17,92)
370 CALL HCHAR(12,13,95):: CALL HCHAR(12,21,95)
380 RESTORE 1380 :: FOR I=1 TO 10 :: READ S,Z,C,L :: CALL HCHAR(S,Z,C,L):: NEXT
I
390 CALL VCHAR(4,13,101,2):: CALL VCHAR(4,21,103,2)
400 W=8
410 CALL HCHAR(12,W,96):: CALL HCHAR(12,W+1,99):: CALL HCHAR(13,W,101):: CALL HC
HAR(13,W+1,103):: CALL HCHAR(14,W,97)
420 CALL HCHAR(14,W+1,98):: W=W+17 :: IF W<28 THEN 410
```

```
430 CALL HCHAR(20,6,104):: CALL HCHAR(20,7,108,2):: CALL HCHAR(20,9,107):: CALL
HCHAR(21,6,109)
440 CALL HCHAR(21,9,111):: CALL HCHAR(22,6,105):: CALL HCHAR(22,7,110,2):: CALL
HCHAR(22,9,106)
450 CALL HCHAR(20,25,104):: CALL HCHAR(20,26,108,2):: CALL HCHAR(20,28,107):: CA
LL HCHAR(21,25,112)
460 CALL HCHAR(21,28,113):: CALL HCHAR(22,25,105):: CALL HCHAR(22,26,110,2):: CA
LL HCHAR(22,28,106)
470 CALL COLOR(2,5,2,8,5,2,9,11,2,10,9,2,11,9,2,3,14,2,4,14,2,5,11,2,14,8,2)
480 CALL CHAR(60,ZE$(19),61,ZE$(20),62,ZE$(1),63,ZE$(18))
490 CALL CHAR(64,ZE$(16),65,ZE$(15),66,ZE$(12),67,ZE$(9),68,ZE$(26),69,ZE$(5))
500 DISPLAY AT(4,12)SIZE(7):"@ABCDEC" :: DISPLAY AT(4,5)SIZE(3):"1:;" :: DISPLAY
 AT(5,3)SIZE(6):"000000" :: DISPLAY AT(4,24)SIZE(3):"2:;"
510 DISPLAY AT(5,22)SIZE(5):"00000" :: CALL HCHAR(5,29,48)
520 WA(1)=32 :: WA(2)=136
530 REM SPIELROUTINEN
540 CALL SPRITE(#1,120,4,81,193):: Y1,Y3=81 :: X1=193 :: X3=57 :: CALL SPRITE(#2
,120,8,Y3,X3)
550 RESTORE 1500 :: FOR SE=3 TO 18 :: READ CH,F,PS,PZ :: CALL SPRITE(#SE,CH,F,PS
,PZ):: NEXT SE
560 CALL CHAR(136,"FFFF00000000FFFF"):: CALL HCHAR(8,7,136):: CALL HCHAR(8,27,13
6):: CALL HCHAR(16,8,136):: CALL HCHAR(16,26,136)
570 CALL HCHAR(18,6,136):: CALL HCHAR(18,28,136)
580 IF NE=1 THEN 590 ELSE 630
590 AB=11 :: DISPLAY AT(7,11)SIZE(5):"<=>?="
600 AB=AB-1 :: FOR ZT=1 TO 50 :: NEXT ZT :: DISPLAY AT(7,17)SIZE(2)BEEP:USING "#
#":AB
610 IF AB=0 THEN DISPLAY AT(7,11)SIZE(8):"        " ELSE 600
620 CALL SOUND(1000,698,1)
630 CALL COINC(ALL,J):: IF J=-1 THEN 910
640 ON JO+1 GOTO 680,650
650 CALL JOYST(1,X,Y):: CALL JOYST(2,X2,Y2):: IF (X=0 AND Y=0)THEN 790
660 ON X+5 GOTO 740,670,670,670,670,670,670,670,700
670 ON Y+5 GOTO 720,660,660,660,790,660,660,660,760
680 CALL KEY(1,S,T):: CALL KEY(2,S1,T1):: IF T=0 THEN 810
690 ON S+1 GOTO 720,810,740,700,810,760
700 X1=X1+8 :: CALL GCHAR(((Y1-1)/8)+1,((X1-1)/8)+1,Z):: IF (Z=32 OR Z=109)OR Z=
113 THEN CALL LOCATE(#1,Y1,X1)ELSE X1=X1-8
710 CALL PATTERN(#1,121):: ON JO+1 GOTO 810,790
720 Y1=Y1+8 :: CALL GCHAR(((Y1-1)/8)+1,((X1-1)/8)+1,Z):: IF (Z=32 OR Z=WA(1))OR(
Z=109 OR Z=113)THEN CALL LOCATE(#1,Y1,X1)ELSE Y1=Y1-8
730 CALL PATTERN(#1,120):: ON JO+1 GOTO 810,790
740 X1=X1-8 :: CALL GCHAR(((Y1-1)/8)+1,((X1-1)/8)+1,Z):: IF (Z=32 OR Z=109)OR Z=
113 THEN CALL LOCATE(#1,Y1,X1)ELSE X1=X1+8
750 CALL PATTERN(#1,122):: ON JO+1 GOTO 810,790
760 Y1=Y1-8 :: CALL GCHAR(((Y1-1)/8)+1,((X1-1)/8)+1,Z):: IF (Z=32 OR Z=WA(1))OR(
Z=109 OR Z=113)THEN CALL LOCATE(#1,Y1,X1)ELSE Y1=Y1+8
770 CALL PATTERN(#1,120):: ON JO+1 GOTO 810,790
790 ON X2+5 GOTO 870,800,800,800,800,800,800,800,830
800 ON Y2+5 GOTO 850,630,630,630,630,630,630,630,890
810 IF T1=0 THEN 630
820 ON S1+1 GOTO 850,680,870,830,680,890
830 X3=X3+8 :: CALL GCHAR(((Y3-1)/8)+1,((X3-1)/8)+1,Z):: IF (Z=32 OR Z=109)OR Z=
113 THEN CALL LOCATE(#2,Y3,X3)ELSE X3=X3-8
840 CALL PATTERN(#2,121):: GOTO 630
850 Y3=Y3+8 :: CALL GCHAR(((Y3-1)/8)+1,((X3-1)/8)+1,Z):: IF (Z=32 OR Z=WA(2))OR(
Z=113 OR Z=109)THEN CALL LOCATE(#2,Y3,X3)ELSE Y3=Y3-8
860 CALL PATTERN(#2,120):: GOTO 630
870 X3=X3-8 :: CALL GCHAR(((Y3-1)/8)+1,((X3-1)/8)+1,Z):: IF (Z=32 OR Z=109)OR Z=
113 THEN CALL LOCATE(#2,Y3,X3)ELSE X3=X3+8
880 CALL PATTERN(#2,122):: GOTO 630
890 Y3=Y3-8 :: CALL GCHAR(((Y3-1)/8)+1,((X3-1)/8)+1,Z):: IF (Z=32 OR Z=WA(2))OR(
Z=113 OR Z=109)THEN CALL LOCATE(#2,Y3,X3)ELSE Y3=Y3+8
```

```
900 CALL PATTERN(#2,120):: GOTO 630
910 CALL COINC(#1,#2,2,FI):: IF FI=-1 THEN 920 ELSE 950
920 MU=0
930 MU=MU+250 :: FOR I=110 TO 550 STEP 100 :: CALL SOUND(100,MU+I,3,MU+I+25,5,MU
+I+50,10):: NEXT I
940 CALL DELSPRITE(#1):: PL=PL+1 :: WE=0 :: WA(1)=136 :: WA(2)=32 :: IF PL>=2 TH
EN 1080 ELSE 540
950 FOR NU=15 TO 18 :: CALL COINC(#(PL+1),#NU,3,NR):: IF NR=-1 THEN CALL DELSPRI
TE(#NU)ELSE 970
960 CALL SOUND(200,-5,1):: PA=INT(RND*3)+3 :: GOSUB 1070 :: GOTO 650
970 NEXT NU
980 FOR GB=11 TO 14 :: CALL COINC(#(PL+1),#GB,2,GR):: IF GR=-1 THEN CALL DELSPRI
TE(#GB)ELSE 1000
990 CALL SOUND(200,-6,1):: PA=INT(RND*11)+11 :: GOSUB 1070 :: GOTO 650
1000 NEXT GB
1010 FOR PE=7 TO 11 :: CALL COINC(#(PL+1),#PE,2,PR):: IF PR=-1 THEN CALL DELSPRI
TE(#PE)ELSE 1030
1020 CALL SOUND(200,-6,1):: PA=INT(RND*7)+7 :: GOSUB 1070 :: GOTO 650
1030 NEXT PE
1040 FOR DU=3 TO 7 :: CALL COINC(#(PL+1),#DU,2,DR):: IF DR=-1 THEN CALL DELSPRIT
E(#DU)ELSE 1060
1050 CALL SOUND(200,-5,1):: PA=INT(RND*60)+60 :: GOSUB 1070 :: GOTO 650
1060 NEXT DU :: GOTO 650
1070 PU(PL+1)=(PU(PL+1))+PA :: DISPLAY AT(5,(3+(PL*19)))SIZE(5):USING "#####":PU
(PL+1):: RETURN
1080 CALL DELSPRITE(ALL):: FOR CL=2 TO 14 :: CALL COLOR(CL,2,2):: NEXT CL
1090 CALL HCHAR(8,7,32):: CALL HCHAR(8,27,32):: CALL HCHAR(16,8,32):: CALL HCHAR
(16,26,32):: CALL HCHAR(18,6,32):: CALL HCHAR(18,28,32)
1100 FOR SC=1 TO 16 :: CALL CHAR(SC+71,ZE$(SC)):: NEXT SC :: CALL CHAR(128,ZA$(2
),129,ZA$(3),130,ZE$(21))
1110 CALL CHAR(137,"0000001818001818")
1120 CALL CHAR(73,ZE$(19),72,ZE$(18)):: CALL COLOR(6,16,2,7,16,2,13,14,2,14,16,2
):: DISPLAY AT(9,12)SIZE(7):"IPLNLH "
1130 CALL HCHAR(9,20,137)
1140 CALL CHAR(82,ZE$(16)):: DISPLAY AT(11,11)SIZE(8):"IRPLSLH "
1150 CALL CHAR(75,"000C18303030190C",77,"0006060C1830606",136,"0030180C0C0C183",
87,"003C66060C180018")
1160 IF PU(1)>PU(2)THEN CALL HCHAR(11,21,128)ELSE IF PU(1)<PU(2)THEN CALL HCHAR(
11,21,129)ELSE CALL HCHAR(11,21,130)
1170 CALL COLOR(12,5,2):: DISPLAY AT(17,3)SIZE(25):"( UVJO LPU IRPLSW KQMU  (" ::
: CALL HCHAR(17,27,136)
1180 CALL KEY(2,CO,ST):: IF CO=2 THEN 1200 ELSE IF CO=15 THEN 1220
1190 GOTO 1180
1200 PL=0 :: PU(1),PU(2)=0 :: DISPLAY AT(17,2)SIZE(26):"                          "
1210 DISPLAY AT(9,12)SIZE(7):"       " :: DISPLAY AT(11,11)SIZE(9):"         "
: WE=1 :: GOTO 470
1220 CALL CLEAR :: STOP
1230 DATA 1,3,29,3,5,6,3,24,6,6,5,6,6,24,6,8,5,6,8,16,3,8,24,6,10,3,8,10,14,2
1240 DATA 10,19,2,10,24,8,14,12,1,14,22,1,16,5,10,16,20,10,18,5,4,18,14,7,18,26,
4
1250 DATA 20,14,1,20,20,1,22,12,3,22,20,3,24,3,29
1260 DATA 2,2,22,4,4,2,13,4,3,19,4,3,11,6,3,4,11,2,11,11,3,19,11,3,9,13,1,13,15,
3
1270 DATA 13,17,3,13,19,3,9,21,1,4,23,2,11,23,3,19,23,3,11,28,3,4,30,2,13,30,3
1280 DATA 19,30,3,2,32,22,19,17,3
1290 DATA 8,13,42,8,21,42,12,15,42,12,17,42,12,19,42,12,4,42,12,30,42,18,11,42
1300 DATA 18,23,42,8,4,43,8,15,43,8,23,43,10,18,43,14,21,43,18,13,43,18,25,43
1310 DATA 20,13,43,20,19,43,22,19,43,14,6,44,14,28,44,16,17,44,22,4,44,22,17,44
1320 DATA 22,30,44,8,11,45,8,19,45,8,30,45,10,16,45,14,13,45,18,9,45,18,21,45
1330 DATA 20,15,45,20,21,45,22,15,45
1340 DATA 1,2,46,3,4,46,3,23,46,10,23,46,18,4,46,6,4,47,6,23,47,10,13,47,14,11,4
7
1350 DATA 16,4,47,16,19,47,22,11,47,24,2,47,6,11,88,6,30,88,10,21,88,14,23,88
```

```
1360 DATA 16,15,88,16,30,88,22,23,88,24,32,88,3,11,89,3,30,89,10,11,89,18,30,89
1370 DATA 1,32,89
1380 DATA 3,13,96,1,6,13,97,1,6,14,102,2,6,16,96,1,6,17,100,1,6,18,99,1,6,19,102
,2
1390 DATA 6,21,98,1,3,21,99,1,3,14,100,7
1400 DATA 00183C667E666666,007C66667C66667C,003C66606060663C,00786C6666666C78
1410 DATA 007E60607860607E,007E60607860606 0,003C66606C66663C,006666667E666666
1420 DATA 003C18181818183C,00060606060666 3C,00666C6C786C6C66,006060606060607E
1430 DATA 00667E7E66666666,006676766E6E6666,003C66666666663C,007C66667C606060
1440 DATA 003C666666766E3C,007C66667C786C66,003C66603C06663C,007E181818181818
1450 DATA 006666666666663C,006666666666 3C18,006666666667E7E24,006666 3C183C6666
1460 DATA 006666663C181818,007E060C1830607E
1470 DATA 003C666E76666 63C,001818381818187E,003C66060C30607E,003C66060C06663C
1480 DATA 000C1C1C2C4C7C0C,007E607C06066 63C,003C66607C66663C,007E660C18181818
1490 DATA 003C66663C66663C,003C66663E06663C
1500 DATA 124,11,161,50,124,11,161,58,124,11,161,200,124,11,161,209
1510 DATA 125,14,81,33,125,8,177,153,125,12,97,185,125,10,49,193,126,11,9,49
1520 DATA 126,11,25,241,126,11,81,89,126,11,145,105,127,13,177,49
1530 DATA 127,13,129,81,127,13,129,177,127,13,65,33
```

Leserecke + Leserecke + Leserecke + Leserecke

Programme für Commodore 16

Ich bin ein neuer Kunde Eurer spitzen Zeitschrift!
Ich finde sie prima, hauptsächlich die vielen Programme.

Nun meine Bitte:
Könntet Ihr auch Programme für den Commodore 16 abdrucken. Ihr würdet mir und anderen Besitzern dieses Computers eine große Freude bereiten.

Rolf Härer, Stuttgart

Redaktion

Es ist prima, daß Ihnen unsere Zeitschrift so gut gefällt.
Zu Ihrer Bitte müssen wir Ihnen sagen, daß der Commodore 16 Computer nicht sehr weit verbreitet ist und somit natürlich auch die entsprechende Software fehlt.
Wir werden jedoch in unserer Zeitschrift „Compute mit" regelmäßig Programme für den Commodore 16 veröffentlichen und auch somit den Besitzern dieses Computersystems entgegenkommen.

Autorenhonorar

In der letzten Ausgabe Ihrer Zeitschrift COMPUTRONIC habe ich gelesen, daß man Ihnen Programmlistings zusenden kann. Allerdings wurde mir nicht ganz klar, wie hoch Programme honoriert werden, da Sie zwei Angaben machten.
Zunächst gaben Sie an, daß jede Seite mit 120 DM honoriert wird. An anderer Stelle erklärten Sie jedoch, daß für das ganze Programm bis zu 120 DM bezahlt wird.

Ich bitte Sie nun mir zu schreiben, welche der beiden Angaben richtig ist.

Matthias Sachs, Haltern

Redaktion

Gern haben wir von Ihrem Brief Kenntnis genommen für uns als Software-Autor tätig zu werden.
Zu dem Honorar möchten wir bemerken, daß wir pro voll abgedruckter Seite 120 DM an die Autoren zahlen. Wir hoffen, Sie sind mit diesem Betrag einverstanden und freuen uns auf eine gemeinsame Zusammenarbeit.

Spitze

Ein Lob an Eure Zeitschriften: Ich finde es Spitze, daß Sie den Abonnenten immer die Zeitschriften Homecomputer und Computronic zusenden, denn ich finde beide Zeitschriften optimal.

Aber etwas Negatives habe ich noch:

Es wäre schön, wenn Sie mehr Anwenderprogramme für den TI-99/4 a Computer veröffentlichen würden.

Bernd Winkler, Freiburg

Redaktion

Vielen Dank für das Lob unserer Zeitschriften. Auch die Redaktion ist erfreut, daß eine Vielzahl der Abonnenten mit dem Kombinations-Abo Computronic und Homecomputer einverstanden sind und diese Entscheidung unseres Verlages so positiv bewerten.
Zu Ihrem Problem mit den Anwenderprogrammen für den TI-99/4: Anwenderprogramme und Informationen für die fortgeschrittenen Programmierer werden in der Zeitschrift Computing Programming Using CPU veröffentlicht. Diese Zeitschrift erhielt ein neues Layout und ist das künftige Insidermagazin für die Programmierfreaks.

Fortsetzung von S. 56

```
2130 DATA 80C0C0C08080808C,,,,,0000000001030708A,153F4792BB924438
2140 DATA 1C2E5FFBD7AEFCF8,F0A0408,,,,,0303030303031B1B1B
2150 DATA 1B1B1F0F03030303,00000C0C6C6C6C7C,7860606060E0C,,,,
2160 DATA 0001030303070703,010103070F1C70C,E0F058F818F0E0F,F8F0E0C,FBFBFB00DFDFD
F,,
2170 DATA ,C020108945227FFF,FF7F2245891020C,438488102040F8FC,FCF8402010888443,FB
FBFB00DFDFDF,
2180 DATA ,,0003070363110907,03070F1F02040818,0080C0808C1020C,80C0E0F08040203,FB
FBFB00DFDFDF
2190 DATA F8F81818F8F81818,,,04040F142C54ABD7,AF8B5C2B6BF7F763,8080C0A0D0A854AC,
D444E85458BCBC18
2200 RESTORE 2040
2210 FOR I=32 TO 143 :: READ C$ :: CALL CHAR(I,C$):: NEXT I
2220 SUBEND
```

mit einem ABO wäre das ... nicht passiert!!!

Wir liefern Ihnen frei Haus:

6 Ausgaben der Zeitschrift „Computronic" für 35,- DM

Computronic

Abonnement

Bitte ausschneiden und senden an:
Tronic-Verlag, Postfach 41, 3444 Wehretal 1.

Hiermit bestelle ich die Zeitschrift »Computronic« ab Heft Nr. _____ zum Jahresabonnementpreis (6 Ausgaben) von DM 35,– inkl. Versand, Inland und DM 45,– inkl. Versand, Ausland.

Name/Vorname: _____

Straße, Nr. _____ Plz., Ort: _____

Ich wünsche folgende Zahlungsweise

☐ Bargeldlos durch Bankeinzug: _____ Bankleitzahl: _____

Geldinstitut: _____ Konto-Nr.: _____

☐ gegen Rechnung

☐ gegen Vorkasse

(Betreffendes bitte ankreuzen) Datum, Unterschrift _____

Abonnements-Kündigungen:
6 Wochen vor Ablauf des Jahres-Abonnements.
Außerdem kann ich diese Bestellung innerhalb von 10 Tagen beim Verlag widerrufen! Zur Wahrung der Frist genügt die rechtzeitige Absendung des Widerrufes.

Datum, Unterschrift _____

RAKETEN-BESCHUSS

TI-99/4A mit EXTENDED BASIC

Steuern Sie Ihre Rakete durch den Asteroidengürtel und vernichten Sie die gegnerischen Raumschiffe

Nach dem Start mit „RUN" erscheint der Name des Spieles und der Sternenhintergrund wird gezeichnet. Es erscheint eine Startrampe im unteren Teil des Bildschirms, die mit dem Joystick Nr. 1 nach links und rechts bewegt werden kann. Am oberen Rand des Bildschirms bewegen sich 6 gegnerische Raumschiffe in 3er Gruppen und beschießen Ihre Rakete mit Bomben. In der Mitte des Bildschirms ziehen in der ersten Runde 4 Asteroiden ihre Bahn. Die Anzahl der Asteroiden steigert sich bis zur 4. Runde bis auf 10 an. In jeder Runde wird die Geschwindigkeit der Asteroiden erhöht. Man muß nun versuchen, mit seiner Rakete eines der gegnerischen Raumschiffe am oberen Bildschirmrand zu treffen, wobei das getroffene Raumschiff dann vernichtet wird. Um zu starten, muß die Feuertaste des Joysticks 1 gedrückt werden. Bei gedrücktem Feuerknopf fliegt Ihre Rakete schneller. Ihr Flug kann mit dem Steuerknüppel nach links oder rechts beeinflußt werden. Aber nach dem Start gibt es kein zurück mehr, denn Ihr Raumschiff hat keine Bremstriebwerke. Berührt man einen der Asteroiden oder eine Bombe der 6 gegnerischen Raumschiffe, wird Ihre Rakete vernichtet und ein Leben abgezogen. Am Anfang stehen Ihnen 3 Leben zur Verfügung. Sie verlieren aber auch dann ein Leben, wenn Ihr Bonus, der unter den anderen Anzeigen zu finden ist, auf Null abgesunken ist. Ein neues Leben erhalten Sie immer, wenn Sie 2 Runden überstanden haben. Haben Sie alle Leben verloren, erscheint „GAME OVER" blinkend in der Mitte des Bildschirms. Zum Start eines neuen Spiels müssen Sie die Taste „j" drücken. Gesteuert wird das Programm mit dem Joystick!

PUNKTEVERTEILUNG:
Pro Treffer: 10 Punkte – Runde
Nach jeder Runde: den Bonus, der 100 – Runde am Anfang der Runde betrug und der pro gegnerischem Schuß um 1 – Runde erniedrigt wurde.

```
100 !@P-
110 !~~~
```

```
273F272725253F7FFFFF000000F888C8C8F8C8C84848F8FCFEFE")! RAKETE AUF STARTRAMPE
370 CALL CHAR(96,"0000000103070707070705050000000000000000080C0C0C0C0C0404000000
00")! RAKETE IM FLUG NORMAL
380 CALL CHAR(100,"0000000103070707070705050 70F152A000000000080C0C0C0C0C04040C0E05
0A8")! RAKETE IM FLUG & ANTRIEB
390 CALL CHAR(132,"0000071F3F7FFFFF7F3F1F07010100000000E0F8FCFEFFFFFEFCF8E080800
000")! GEGNER
400 CALL CHAR(136,"01071F3F7F7FFFFF7F7F3F1F07010000 9ABDB5DBBD5DF55ADBBDD55AA5AB0
000")! ASTEROID-FLUG NACH LINKS
410 CALL CHAR(140,"9ABDB5DBBD5DF55ADBBDD55AA5AB000080E0F8FCFEFEFFFFFEFEFCF8E0800
000")! ASTEROID-FLUG NACH RECHTS
420 CALL CHAR(124,"0000000001030 70F0F0F070300000000000000000080C0E0F0F0F0E0C000000
000")! BOMBE
430 CALL CHAR(104,"00000105 0A1D2B5D5D2B1D0B050100000000802010A8BCA295DEBCA840800
000")! EXPLOSIONSWOLKE 1
440 CALL CHAR(108,"01050A1D2B5DDAA9DBBD5D2B1D0B05018 02010A8BCDEBBDBA295DEBCA8102
080")! EXPLOSIONSWOLKE 2
450 CALL CHAR(119,"0000000000000FF",120,"10",121,"00001818",122,"01000000000000
80",123,"0000000000000303")! BODEN & STERNE
460 ! *** STERNE ***
470 FOR I=1 TO 100 :: RANDOMIZE :: CALL PEEK(-31880,R1):: RANDOMIZE :: CALL PEEK
(-31880,R2):: RANDOMIZE :: CALL PEEK(-31880,R3)
480 CALL HCHAR(INT(R1/4.8+3),INT(R2/3.4+2),INT(R3/25)+120):: NEXT I
490 CALL VCHAR(1,1,32,23):: CALL VCHAR(1,32,32,23)
500 CALL HCHAR(24,1,119,32):: CALL HCHAR(1,2,32,30)
510 DISPLAY AT(2,1):"yx (z z(xy yx yy( (z z(yz xz"
520 ! *** SPIELBEGINN ***
530 DISPLAY AT(1,1):"PUNKTE:" :: DISPLAY AT(1,15):"LEVEL:" :: DISPLAY AT(1,24):"
LEB:"
540 LE=0 :: SC=-10 :: L=3
550 LE=LE+1 :: SC=SC+B :: B=100*LE :: IF LE>4 THEN GE=GE-1
560 SC=SC+90*(LE-1):: IF LE=INT(LE/2)*2 THEN L=L+1
570 A(1),A(2),A(3),A(4),A(5),A(6)=0 :: FOR I=40 TO 200 STEP 80 :: CALL SPRITE(#(
I+40)/80+3,132,16,20,I,0,-6):: NEXT I
580 FOR I=200 TO 40 STEP -80 :: CALL SPRITE(#(I+40)/80,132,16,40,I,0,6):: NEXT I
590 CALL SPRITE(#7,128,16,176,128)
600 FOR I=1 TO 1+LE+GE :: RANDOMIZE :: CALL PEEK(-31880,R1):: RANDOMIZE :: CALL
PEEK(-31880,R2):: CALL SPRITE(#I+7,140,16,60+I*20,20+R1,0,R2/20+LE)
610 NEXT I
620 FOR I=1 TO 1+LE+GE :: RANDOMIZE :: CALL PEEK(-31880,R1):: RANDOMIZE :: CALL
PEEK(-31880,R2):: CALL SPRITE(#I+17,136,16,60+I*20,155+R1,0,-(R2/20+LE))
630 NEXT I
640 S(1),S(2),S(3)=0
650 CALL MOTION(#7,0,0):: CALL LOCATE(#7,176,128):: CALL PATTERN(#7,128)
660 SC=SC+10*LE :: DISPLAY AT(1,8):USING "#####":SC):: DISPLAY AT(1,21):USING "#
#":LE):: DISPLAY AT(1,28):USING "#":L
670 ! *** ALS STARTRAMPE ***
680 RANDOMIZE :: CALL PEEK(-31880,R):: IF R>80 THEN CALL S(S(),A(),B,LE)ELSE RAN
DOMIZE :: CALL PEEK(-31880,R):: CALL COLOR(12,INT(R/7.2+3),2):: DISPLAY AT(3,12)
:B
690 CALL SOUND(-4250,-8,5)
700 CALL JOYST(1,X,Y):: CALL MOTION(#7,0,X/2):: CALL KEY(1,X,Y):: IF X=18 THEN C
ALL PATTERN(#7,100):: CALL MOTION(#7,-4,0):: GOTO 760
710 IF S(1)=1 THEN CALL COINC(#7,#26,10,C):: IF C THEN CALL E(L):: GOTO 590 ELSE
 CALL POSITION(#26,X,Y):: IF X>180 THEN CALL DELSPRITE(#26):: S(1)=0
720 IF S(2)=1 THEN CALL COINC(#7,#27,10,C):: IF C THEN CALL E(L):: GOTO 590 ELSE
 CALL POSITION(#27,X,Y):: IF X>180 THEN CALL DELSPRITE(#27):: S(2)=0
730 IF S(3)=1 THEN CALL COINC(#7,#28,10,C):: IF C THEN CALL E(L):: GOTO 590 ELSE
 CALL POSITION(#28,X,Y):: IF X>180 THEN CALL DELSPRITE(#28):: S(3)=0
740 GOTO 680
750 ! *** START ***
760 RANDOMIZE :: CALL PEEK(-31880,R):: IF R>80 THEN CALL S(S(),A(),B,LE)ELSE RAN
```

```
DOMIZE :: CALL PEEK(-31880,R):: CALL COLOR(12,INT(R/7.2+3),2):: DISPLAY AT(3,12)
:B
770 CALL PATTERN(#7,96):: CALL SOUND(-4250,-5,10)
780 IF S(1)=1 THEN CALL COINC(#7,#26,10,C):: IF C THEN CALL E(L):: GOTO 590 ELSE
 CALL POSITION(#26,X,Y):: IF X>180 THEN CALL DELSPRITE(#26):: S(1)=0
790 IF S(2)=1 THEN CALL COINC(#7,#27,10,C):: IF C THEN CALL E(L):: GOTO 590 ELSE
 CALL POSITION(#27,X,Y):: IF X>180 THEN CALL DELSPRITE(#27):: S(2)=0
800 IF S(3)=1 THEN CALL COINC(#7,#28,10,C):: IF C THEN CALL E(L):: GOTO 590 ELSE
 CALL POSITION(#28,X,Y):: IF X>180 THEN CALL DELSPRITE(#28):: S(3)=0
810 CALL KEY(1,K,O):: IF K=18 THEN CALL PATTERN(#7,100):: CALL JOYST(1,X,Y):: CA
LL MOTION(#7,-5,X/2)ELSE CALL JOYST(1,X,Y):: CALL MOTION(#7,-1,X/4)
820 CALL POSITION(#7,K,O):: IF K<60 THEN 870
830 FOR I=8 TO 8+LE :: CALL COINC(#7,#I,14,C):: IF C THEN CALL E(L):: GOTO 590
840 NEXT I
850 FOR I=18 TO 18+LE :: CALL COINC(#7,#I,14,C):: IF C THEN CALL E(L):: GOTO 590
860 NEXT I :: GOTO 760
870 FOR I=1 TO 6 :: CALL COINC(#I,#7,16,C):: IF C THEN CALL T(I,A(),AN):: IF AN=
0 THEN 550 ELSE 640
880 NEXT I :: GOTO 760
890 !@P+
900 ! *** UNTERPROGRAMM ***        *** BOMBENABWURF  ***
910 SUB S(S(),A(),B,LE)
920 GOTO 930 :: LE,B,R,X,Y,S(1),S(2),S(3),A(1),A(2),A(3),A(4),A(5),A(6):: CALL S
PRITE :: CALL POSITION :: CALL SOUND :: !@P-
930 IF S(1)+S(2)+S(3)=3 THEN SUBEXIT
940 B=B-LE
950 RANDOMIZE :: CALL PEEK(-31880,R)
960 ON INT(R/17+1)GOTO 970,990,1010,1030,1050,1070
970 IF A(1)=1 THEN 990 ELSE CALL POSITION(#1,X,Y):: IF S(1)=0 THEN S(1)=1 :: CAL
L SPRITE(#26,124,16,X,Y,5,0):: SUBEXIT
980 IF S(2)=0 THEN S(2)=1 :: CALL SPRITE(#27,124,16,X,Y,5,0):: SUBEXIT ELSE S(3)
=1 :: CALL SPRITE(#28,124,16,X,Y,5,0):: SUBEXIT
990 IF A(2)=1 THEN 1010 ELSE CALL POSITION(#2,X,Y):: IF S(1)=0 THEN S(1)=1 :: CA
LL SPRITE(#26,124,16,X,Y,5,0):: SUBEXIT
1000 IF S(2)=0 THEN S(2)=1 :: CALL SPRITE(#27,124,16,X,Y,5,0):: SUBEXIT ELSE S(3
)=1 :: CALL SPRITE(#28,124,16,X,Y,5,0):: SUBEXIT
1010 IF A(3)=1 THEN 1030 ELSE CALL POSITION(#3,X,Y):: IF S(1)=0 THEN S(1)=1 :: C
ALL SPRITE(#26,124,16,X,Y,5,0):: SUBEXIT
1020 IF S(2)=0 THEN S(2)=1 :: CALL SPRITE(#27,124,16,X,Y,5,0):: SUBEXIT ELSE S(3
)=1 :: CALL SPRITE(#28,124,16,X,Y,5,0):: SUBEXIT
1030 IF A(4)=1 THEN 1050 ELSE CALL POSITION(#4,X,Y):: IF S(1)=0 THEN S(1)=1 :: C
ALL SPRITE(#26,124,16,X,Y,5,0):: SUBEXIT
1040 IF S(2)=0 THEN S(2)=1 :: CALL SPRITE(#27,124,16,X,Y,5,0):: SUBEXIT ELSE S(3
)=1 :: CALL SPRITE(#28,124,16,X,Y,5,0):: SUBEXIT
1050 IF A(5)=1 THEN 1070 ELSE CALL POSITION(#5,X,Y):: IF S(1)=0 THEN S(1)=1 :: C
ALL SPRITE(#26,124,16,X,Y,5,0):: SUBEXIT
1060 IF S(2)=0 THEN S(2)=1 :: CALL SPRITE(#27,124,16,X,Y,5,0):: SUBEXIT ELSE S(3
)=1 :: CALL SPRITE(#28,124,16,X,Y,5,0):: SUBEXIT
1070 IF A(6)=1 THEN 970 ELSE CALL POSITION(#6,X,Y):: IF S(1)=0 THEN S(1)=1 :: CA
LL SPRITE(#26,124,16,X,Y,5,0):: SUBEXIT
1080 IF S(2)=0 THEN S(2)=1 :: CALL SPRITE(#27,124,16,X,Y,5,0):: SUBEXIT ELSE S(3
)=1 :: CALL SPRITE(#28,124,16,X,Y,5,0):: SUBEXIT
1090 !@P+
1100 SUBEND
1110 ! *** UNTERPROGRAMM ***        ***   GETROFFEN   ***
1120 SUB E(L)
1130 GOTO 1140 :: L :: CALL KEY :: CALL COLOR :: CALL MOTION :: CALL DELSPRITE :
: CALL LOCATE :: CALL PATTERN :: A,B,K,S=0 :: !@P-
1140 CALL MOTION(#7,0,0):: CALL PATTERN(#7,104):: CALL SOUND(-400,-7,0):: CALL P
AUSE :: CALL PATTERN(#7,108):: CALL DELSPRITE(#26,#27,#28)
1150 CALL SOUND(-400,-6,0):: CALL PAUSE :: CALL LOCATE(#7,176,128):: CALL PATTER
N(#7,128):: CALL SOUND(1,30000,30)
```

```
1160 L=L-1 :: IF L>0 THEN SUBEXIT
1170 ! *** LETZTES LEBEN ***        ***   VERLOREN    ***
1180 A=INT(RND*14+3):: B=INT(RND*14+3):: CALL COLOR(5,A,2,6,A,2,7,A,2,8,A,2,12,I
NT(RND*14+3),2,3,B,2,4,B,2)
1190 DISPLAY AT(12,5)BEEP:"G A M E  O V E R" :: DISPLAY AT(12,5):"x z y  (  y xz
 ("
1200 CALL KEY(0,K,S):: IF K=106 THEN RUN 290 ELSE 1180
1210 !@P+
1220 SUBEND
1230 ! ***    GEGNER    ***        ***   ERREICHT   ***
1240 SUB T(I,A(),AN)
1250 GOTO 1260 :: CALL DELSPRITE :: CALL SOUND :: I,AN,A(1),A(2),A(3),A(4),A(5),
A(6)=0 :: !@P-
1260 CALL SOUND(-1000,-6,0)
1270 CALL DELSPRITE(#I):: A(I)=1 :: AN=AN+1 :: IF AN<6 THEN CALL DELSPRITE(#26,#
27,#28):: SUBEXIT
1280 AN=0 :: CALL DELSPRITE(ALL)
1290 !@P+
1300 SUBEND
1310 ! *** UPG. PAUSE ***
1320 SUB PAUSE
1330 I=0 :: !@P-
1340 FOR I=1 TO 100 :: NEXT I
1350 !@P+
1360 SUBEND
1370 ! *** UPG. TITEL ***
1380 SUB TITEL
1390 CALL VCHAR(1,1,42,24):: CALL VCHAR(1,32,42,24):: CALL HCHAR(1,2,42,30):: CA
LL HCHAR(24,2,42,30)
1400 CALL COLOR(2,16,2,11,16,2,12,16,2,5,16,2,6,16,2,7,16,2,8,16,2,3,16,2,4,16,2
)
1410 DISPLAY AT(24,7):"RAKETEN-BESCHUSS":: DISPLAY AT(1,8):"(C)1984/85 BY":: D
ISPLAY AT(2,7):"THOMAS GOERLICH"
1420 SUBEND
```

Spannendes Flugzeugduell um die Luftherrschaft

Als Führer eines bewaffneten Kampfflugzeugs haben Sie den Auftrag erhalten, in Ihrem Einsatzgebiet die Luftherrschaft wiederzugewinnen.

Nach Ihrem Start dringen Sie in den von feindlichen Flugzeugen beherrschten Luftraum ein. Plötzlich werden Sie von Ihren Gegnern angegriffen. Schießen Sie den Feind ab, bevor Sie getroffen werden!

Gesteuert wird Ihr Flugzeug mit den folgenden Tasten:

S – Flugzeug fliegt nach links
D – Flugzeug fliegt nach rechts
L – Auslösen eines Schusses
Die „Alpha-Lock"-Taste muß gedrückt sein.

Sie können zwischen Tag- und Nachtflug wählen. Beim Nachtflug sehen Sie Ihren Gegner nicht. Sie können ihn jedoch durch einen Peilton orten. Wenn ein hoher Ton erklingt, befindet sich das feindliche Flugzeug rechts von Ihnen, bei einem tiefen Ton links von Ihnen. Hören Sie einen mehrstimmigen Ton, dann haben Sie Ihren Gegner genau vor sich. Dies ist der richtige Moment zum Schießen! Wenn Ihr Rivale zurückschießt, ist Ihre Feuertaste gesperrt. Sie müssen dann dem Schuß ausweichen, um einem Treffer zu entgehen.

```
10 RANDOMIZE
20 CALL CLEAR :: CALL CHAR(36,"FFFFFFFFFFFFFFFF"):: CALL CHAR(35,"FFFFFFFFFFFFFF
FF")
30 CALL CHAR(40,"0000000000001070A0D070A0D0300000000000000000040A050E0B0D0604000000
A")
40 DATA 0000000003070F0F0F0F0703000000000000000000C0E0F0F0F0F0E0C000000000,0000000
1031F3F7FFF7FFF7F1F0B00000000E0F0FEFFFFFFFEFCFCFEFCF80000
50 DATA 0000000000000000020202FF0F0906000000000000000000040404 0FFF0906000,0000001
0080402070503010000000000000000000000408010A0C060D008040000
60 DATA 0000000000020108050306 0B102000000000000008102040E0A0C080000000000,0000000
8080808080808080000000000000000010101010101010000000000
70 DATA 0000000000000404040404000000000000000000000000020202020200000000000,0000000
0000000000020202000000000000000000000000000040404000000000000
80 DATA 0000000000000000000001000000000000000000000000000000000800000000000,0000000
0000000002021F030000000000000000000000000004040F8C0000000000
90 DATA 0000000804030301000000000000000000000000000800020C0C0201000000000000,0000000
0010004030304080000000000000000001020C0C080000000000000000
100 FOR I=96 TO 140 STEP 4 :: READ A$ :: CALL CHAR(I,A$):: NEXT I
110 FOR I=3 TO 9 :: CALL COLOR(I,16,2):: NEXT I :: CALL SCREEN(2):: CALL DELSPRI
TE(ALL):: CALL MAGNIFY(1):: CALL COLOR(1,12,2)
120 DISPLAY AT(24,1):"" :: DISPLAY AT(5,1):"TAG- ODER NACHTFLUG [T/N] ?"
130 ACCEPT AT(10,4)VALIDATE(NUMERIC,"TN")BEEP:B$ :: IF B$="N" THEN GOSUB 630 ELS
E GOSUB 620
140 DISPLAY AT(16,4):"SCHWIERIGKEITSGRAD [1 8]" :: ACCEPT AT(20,5)VALIDATE(NUMER
IC,"12345678")BEEP:SC :: IF SC>8 THEN 140
150 SC=SC+2 :: GE=SC :: AN=SC*3+7 :: FL=3
160 CALL HCHAR(10,1,32,32)
170 CALL HCHAR(5,1,36,32):: CALL HCHAR(20,1,36,32):: J=50 :: FOR I=1 TO 4 :: CAL
L SPRITE(#I,35,2,33,J,0,-100):: J=J+50 :: NEXT I
180 J=50 :: FOR I=5 TO 8 :: CALL SPRITE(#I,35,2,153,J,0,-100):: J=J+50 :: NEXT I
190 DISPLAY AT(1,3):"PRESS ANY KEY TO PLAY" :: DISPLAY AT(8,8):"F I R E F O X" :
: DISPLAY AT(16,3):"CO.1985 BY: DIRK BLUDAU"
200 DISPLAY AT(5,2)SIZE(27):"INTERFACE SOFTWARE PRESENTS"
210 FOR I=1 TO 3 :: FOR SD=200 TO 550 STEP 20 :: CALL SOUND(55,SD,0,SD+15,0,500,
0):: NEXT SD :: NEXT I
220 CALL KEY(0,K,S):: IF S=0 THEN 220 :: CALL CLEAR :: DISPLAY AT(24,1)SIZE(11):
"MUNITION:" :: DISPLAY AT(24,10)SIZE(3):AN
230 DISPLAY AT(24,20)SIZE(9):"GEGNER:" :: DISPLAY AT(24,27)SIZE(2):FL
240 CALL DELSPRITE(ALL):: CALL MAGNIFY(4):: CALL SCREEN(FA):: CALL SPRITE(#9,96,
AB,1,128)
250 CALL SPRITE(#1,104,GG,155,128):: CALL SPRITE(#3,132,CC,14,128):: CALL COLOR(
1,AF,AF)
260 CALL SPRITE(#4,100,AA,130,10,#5,100,AA,40,60,#6,100,AA,10,120,#7,100,AA,100,
160)
270 CALL SPRITE(#8,100,AA,70,210)
280 FOR I=4 TO 8 :: CALL MOTION(#I,50,0):: NEXT I :: CALL MOTION(#9,0,0,#3,0,R):
: CALL PATTERN(#1,104)
290 IF WW=1 THEN 700
300 IF B$="N" THEN GOSUB 640
310 GOSUB 480
320 CALL KEY(0,K,S):: IF S=0 THEN 290 :: IF K=83 THEN 340 ELSE IF K=68 THEN 390
ELSE IF WW=1 THEN 700 ELSE IF K=76 THEN 440 ELSE 290
330 GOTO 300
340 CALL PATTERN(#1,112):: FOR I=4 TO 8 :: CALL MOTION(#I,20,50):: NEXT I :: CAL
L MOTION(#9,0,15):: R=5 :: IF ZAE=0 THEN R=15
350 CALL MOTION(#3,0,R)
360 IF WW=1 THEN CALL MOTION(#10,15,R):: GOTO 710
370 IF B$="N" THEN GOSUB 640
380 CALL KEY(0,K,S):: IF S=0 THEN 280 ELSE IF K=76 THEN 440 ELSE IF K=68 THEN 39
0 ELSE GOSUB 480 :: GOTO 370
390 CALL PATTERN(#1,108):: FOR I=4 TO 8 :: CALL MOTION(#I,20,-50):: NEXT I :: CA
LL MOTION(#9,0,-15):: R=-5 :: IF ZAE=0 THEN R=-15
```

```
400 CALL MOTION(#3,0,R)
410 IF WW=1 THEN CALL MOTION(#10,15,R):: GOTO 710
420 IF B$="N" THEN GOSUB 640
430 CALL KEY(0,K,S):: IF S=0 THEN 280 ELSE IF K=76 THEN 440 ELSE IF K=83 THEN 34
0 ELSE GOSUB 480 :: GOTO 420
440 AN=AN-1 :: IF AN=0 THEN 590 ELSE DISPLAY AT(24,10)SIZE(3):AN :: CALL SOUND(5
00,-6,0)
450 CALL SPRITE(#2,116,BB,155,128,-17,0):: GOSUB 470 :: CALL PATTERN(#2,120):: G
OSUB 470 :: CALL PATTERN(#2,124):: GOSUB 470
460 CALL PATTERN(#2,128):: CALL COINC(#2,#3,10,T):: IF T=-1 THEN GOTO 530 ELSE C
ALL DELSPRITE(#2):: GOTO 300
470 FOR W=1 TO 160 :: NEXT W :: RETURN
480 IF ZAE=2 THEN O=O+1 :: IF O=SC THEN ZAE,O=0 :: GOTO 490 ELSE RETURN
490 IF ZAE=1 THEN O=O+1 :: IF O=SC THEN ZAE,O=0 :: GOTO 500 ELSE RETURN
500 H=INT(RND*6)+1 :: IF H=1 THEN 660 ELSE IF H<4 THEN 510 ELSE IF H<6 THEN 520
ELSE CALL PATTERN(#3,132):: CALL MOTION(#3,0,0):: RETURN
510 CALL PATTERN(#3,140):: CALL MOTION(#3,0,-20):: ZAE=1 :: RETURN
520 CALL PATTERN(#3,136):: CALL MOTION(#3,0,20):: ZAE=2 :: RETURN
530 IF B$="N" THEN CALL POSITION(#3,U,V):: CALL SPRITE(#3,132,16,U,V)
540 CALL DELSPRITE(#2):: CALL PATTERN(#3,40):: CALL SOUND(1000,-7,0):: GOSUB 700
 :: CALL PATTERN(#3,140):: CALL MOTION(#3,15,-10)
550 CALL POSITION(#3,U,V):: IF U>159 THEN 560 ELSE 550
560 FL=FL-1 :: IF FL=0 THEN 570 ELSE DISPLAY AT(24,27)SIZE(2):FL :: GOTO 240
570 CALL DELSPRITE(#3):: DISPLAY AT(1,8):"GAME WON !!!"
580 FOR I=1 TO 3 :: FOR SD=500 TO 1000 STEP 40 :: CALL SOUND(100,SD,0,SD+10,0,40
0,0):: NEXT SD :: NEXT I :: GOTO 600
590 DISPLAY AT(1,8):"GAME LOST !!!" :: FOR SD=500 TO 200 STEP -10 :: CALL SOUND(
100,SD,0,230,0,SD-10,0):: NEXT SD
600 DISPLAY AT(10,5):"F I R E F O X"
610 DISPLAY AT(19,1):"DO YOU WISH TO PLAY AGAIN ??" :: ACCEPT AT(20,10)VALIDATE(
NUMERIC,"YN")BEEP:Q$ :: IF Q$="Y" THEN 720 ELSE END
620 AA=16 :: BB=7 :: CC=5 :: GG=2 :: FA=6 :: AF=8 :: RETURN
630 AA=5 :: BB=11 :: CC=2 :: GG=15 :: FA=2 :: AF=2 :: RETURN
640 CALL POSITION(#3,U,V):: IF V>132 THEN CALL SOUND(100,500,0)ELSE IF V<124 THE
N CALL SOUND(100,150,0)ELSE CALL SOUND(250,300,0,700,0)
650 RETURN
660 WW=1
670 CALL POSITION(#3,U,V):: CALL SPRITE(#10,128,BB,U,V,35,0):: GOSUB 320 :: CALL
 PATTERN(#10,124):: GOSUB 320 :: CALL PATTERN(#10,120)
680 GOSUB 320 :: CALL PATTERN(#10,116):: CALL COINC(#10,#1,17,L):: IF L=-1 THEN
GOTO 690 ELSE CALL DELSPRITE(#10):: WW=0 :: GOTO 300
690 CALL DELSPRITE(#10):: CALL PATTERN(#1,40):: CALL SOUND(1000,-7,0):: GOSUB 70
0 :: CALL DELSPRITE(#1):: WW=0 :: GOTO 590
700 FOR I=1 TO 45 :: NEXT I :: RETURN
710 FOR I=1 TO 50 :: NEXT I :: RETURN
720 CALL HCHAR(19,1,32,32):: CALL HCHAR(20,1,32,32):: GOTO 110
```

Widerstandsdecodierung

apple

Jeder Hobbyelektroniker weiß um das Problem, einen geeigneten Widerstand anhand seiner Farbcodierung zu ermitteln. Dieses Programm errechnet, nach Eingabe der genormten Widerstandsfarben, den genauen Widerstandswert in einer geeigneten Dimensionierung (Ohm, KOhm, Megaohm). Dabei wird sowohl für die Kohleschichtwiderstände (4 Ringe) als auch für die Metallschichtwiderstände (5 Ringe) eine Berechnung vorgenommen. Darüber hinaus wird der Toleranzring ausgewertet und prozentual ausgedruckt. Anschließend werden die Absolutwerte ausgedruckt, und das Programm verlangt eine neue Eingabe. Falsche Eingaben werden mit einem Rücksprung zum 1. Farbring ausgewertet. Die Widerstandsfarbabkürzungen werden mit einer einleitenden Farbtabelle erklärt. Dieses Programm gibt jedem Hobbyelektroniker die Möglichkeit, mit Hilfe des Computers eine genaue Bestimmung der Widerstände vorzunehmen. Lästige Berechnungen anhand der Farbbezeichnungen entfallen gänzlich.

```
10  REM  * R.DECODIERUNG *
20  REM  * >C< 31.01.85  *
30  REM  * AUTOREN :     *
40  REM  * J.OFFERHAUS   *
50  REM  * A.BITTNER     *
60  DIM A(5)
70  J = 99
```

MINER 99er

Das Spiel MINER 99er ist für den TI-99 mit EXTENDED BASIC und ohne Speichererweiterung geschrieben worden.

MINER wandert durch eine verlassene Mine und sammelt Gegenstände ein. Dabei muß er verschiedenen Monstern ausweichen. Während seiner Wanderung benutzt er Leitern, Rutschen, Fahrstühle und eine Kanone.

Die Idee dieses Spiels wird wahrscheinlich vielen Lesern bekannt sein. Es ist aber kein einfaches „Abkupfern". Das Spiel ist zwar nach dieser Idee gestaltet, ist aber in BASIC geschrieben und völlig anders aufgebaut. MINER 99er läuft über sechs verschiedene Schwierigkeitsstufen. Nach drei völlig verschiedenen Screens werden diese in einer etwas schwereren Stufe wiederholt.

Programmeintippen:

Wer eine Speichererweiterung hat, kann beide Programme als ein Programm abtippen. Man braucht die Zeilen 680 bis 710 aus Listing 1 nicht abtippen.

Für Leute ohne Speichererweiterung: Tippen Sie zuerst Listing Nr. 2 (Hauptprogramm) ab! Gehen Sie aber dabei bitte äußerst sorgfältig vor. Dies ist sehr wichtig, da beim späteren Programmrun mit beiden Programmen jeder Fehler das Hauptprogramm unterbrechen und die Charaktersätze auf die alten normalen Charaktersätze zurückgesetzt würden.

Wir empfehlen das Programm nach 10 eingetippten Zeilen gründlich zu überprüfen und nach jeden 20 Zeilen zu SAVEN. Außerdem sollten Sie vor jeder ON ERROR-Zeile erst mal ein REM setzen, damit Sie beim Versuchsstart normale Fehlermeldungen bekommen. Nachdem Sie das Listing Nr. 2 vollständig abgetippt haben, müssen Sie es starten. Das Bild, das Sie dann auf Ihrem Bildschirm bekommen, wird ein bißchen komisch aussehen, weil die Charakter in Programm 1 definiert werden, das Sie aber erst später abtippen sollen.

Nachdem Sie das Programm gestartet haben, warten Sie, bis das Programm durch mindestens 3 verschiedene Screens gelaufen ist. Dann drücken Sie irgendeinen Joystickbutton, um das Spiel zu starten. Nun bewegen Sie Ihren Joystick, versuchen Sie herumzugehen, die Leitern (noch als C, A, B zu erkennen) hinaufzuklettern und zu springen (mit Joystickbutton). Nur in folgenden Fällen sollten Fehlermeldungen erscheinen:

- wenn sich unter Ihrem Männchen kein Zeichen befindet (z. B. Bildschirmrand)
- wenn Sie einen anderen Sprite berühren

Anschließend SAVEN Sie das Spiel, ohne die REMs vor den ON ERROR-Zeilen.

Jetzt kann Listing Nr. 1 mit RUN „CS1". Nach dem Verklingen der Melodie drücken Sie einmal die ENTER-Taste, starten das Tape (es dürfte jetzt nicht laufen) und drücken ENTER (jetzt wird das Hauptprogramm geladen, die Sprites müßten stillstehen). Nach dem Ladevorgang startet das Programm automatisch. Warten Sie, damit Sie die einzelnen Screens genau betrachten können. Dann drücken Sie irgendeinen Joystickbutton, um das Spiel zu starten. Ihre Aufgabe ist es, alles innerhalb der angezeigten Zeit plattzutreten. Wenn man einen Hammer ergreift (nur durch Springen möglich, der Hammer muß sich dabei genau in der Mitte zwischen Start- und Landepunkt befinden), färben sich die Monster für einen kurzen Zeitraum weiß und es ist möglich, diese dann durch Berühren zu töten. Ist alles plattgetreten, stellen Sie sich unter das END-Zeichen und bewegen Sie den Joystick nach oben, um in den nächsten Screen zu kommen. Im zweiten Screen sind die Fahrstühle neu. Begeben Sie sich genau in die Mitte und bewegen Sie den Joystick nach oben oder unten, um in die nächste Etage transportiert zu werden. Im dritten Screen können Sie sich mit der Kanone in andere Etagen schießen. Stellen Sie sich unter die Munition und bewegen Sie den Joystick nach oben, um die Munition aufzunehmen. Dann wandern Sie in die Kanonenöffnung. Sie können nun die Kanone bewegen und sich mit Hilfe des Joysticks abschießen. Vergessen Sie aber nicht, die Kanone zu laden bzw. zu überladen.

Im 1. Screen ist es möglich, alle vier Monster zu töten. Im 2. kann man drei Monster töten und in allen anderen Screens ist ein Töten der Monster überhaupt nicht möglich.

Also, helfen Sie dem kleinen Mann bei seiner schweren Arbeit im Berg!

```
10 ! ***** MINERSTART *****
20 ! ***** WRITTEN BY *****
30 ! ** H. STAMERJOHANNS **
40 ! ****** NOV 1984 ******
50 CALL CLEAR :: CALL SCREEN(2):: CALL MAGNIFY(2)
60 DATA 77,73,78,69,82,57
70 FOR I=1 TO 6 :: READ A :: CALL CHARPAT(A,A$):: CALL CHAR(135+I,A$):: NEXT I
80 DATA 136,137,138,139,140,141,141,139,140
90 FOR I=1 TO 9 :: READ A :: CALL SPRITE(#I,A,I+5,I*15+10,I*20+20):: NEXT I
100 CALL CHAR(33,"25253D2525000000EA0BCB0AEA0000005E52DED452")
110 CALL CHAR(36,"BDA1A1A1BD000000212 1E12021")
120 CALL CHAR(104,"EE04E424E4000000F496F59594000000 5ED05C505E000000F191F1A597")
130 CALL CHAR(108,"7A4A4B4A7A0000005E52DE5252000000 94D6F7B594000000BCA0BC84BC")
140 CALL COLOR(1,16,2,10,16,2)
150 DISPLAY AT(24,8):"!""#$%hiJklmno"
160 CALL SOUND(250,349,2)
170 CALL SOUND(115,349,2)
180 CALL SOUND(380,349,2,175,8)
190 A$="CC33CCFFFF" :: CALL CHAR(38,"FFFFFFFFFF000000"&A$)
200 CALL SOUND(380,262,2,175,8)
210 CALL CHAR(40,"FFFFFFFFFF81FF81")
220 CALL SOUND(250,440,2,175,8)
230 CALL CHAR(41,A$&"81FF81"):: CALL CHAR(42,"FFFFFFFFFFFFFFFF")
240 CALL SOUND(115,440,2,175,8)
250 CALL SOUND(380,440,2,175,8)
260 CALL CHAR(43,A$&"FFFFFF"):: CALL CHAR(44,"FFFFFFFFFFFFFFFF")
270 CALL SOUND(380,349,2,175,8)
280 CALL CHAR(45,A$&"FFFFFF"):: CALL CHAR(46,"7446774574"):: CALL CHAR(47,"B8A4A
4A4B8")
290 CALL SOUND(250,349,2,175,8)
300 CALL SOUND(115,440,2,175,8)
310 CALL SOUND(500,524,2,175,8)
320 CALL CHAR(58,"00F78484F41414F7",59,"00BD2525252525BD",60,"00EF2828EE48282F",
61,"00EA4B4B4A4A4A4A",62,"002F68E8AE28282F")
330 CALL SOUND(190,524,2,175,8)
340 CALL SOUND(190,466,2,175,8)
350 CALL SOUND(190,440,2,175,8)
360 CALL SOUND(750,392,2,131,8)
370 CALL CHAR(64,"81FF8181FF8181FF8181FF8181FF8181FF8181FF8181FF81")
380 CALL SOUND(250,392,2,131,8)
390 CALL SOUND(115,440,2,131,8)
400 CALL SOUND(380,466,2,131,8)
410 CALL CHAR(68,"0103070F1F3F7FFFFFFEFCF8F0E0C080FF7F3F1F0F07030180C0E0F0F8FCFE
FF")
420 CALL SOUND(380,466,2,131,8)
430 CALL CHAR(72,"E0E0E0E0E0E0E0E0070707070707070707FFFF0000000000000003FFFCFCE0E0E0
E0")
440 CALL SOUND(250,440,2,131,8)
450 CALL CHAR(76,"C0FF3F3F07070707")
460 CALL SOUND(115,392,2,131,8)
470 CALL SOUND(380,440,2,175,8)
480 CALL CHAR(80,"0000000000000000000000018181800000000006666660000001818180066 66
66")
490 CALL SOUND(380,349,2,175,8)
500 CALL CHAR(84,"7446774574000000B8A4A4A4B800000030FC36333030303000002E6A2A2A2E
")
510 CALL SOUND(250,349,2,175,8)
520 CALL SOUND(115,440,2,175,8)
530 CALL SOUND(380,392,2,131,8)
540 CALL CHAR(88,"FF3F3F3F3F3FFF8F8F8F8FE1E1E1E1FFFFFCFCFCFCFCFFF1F1F1F187878787
FF")
```

```
550 CALL SOUND(380,262,2,131,8)
560 CALL CHAR(92,"0F0F0F0F0F0F0F0F0F0F0F0F1F1F3FFFF0F0F0F0F0F0F0F0F0F0F0F8F8FC
FF")
570 CALL SOUND(250,330,2,131,8)
580 CALL SOUND(115,392,2,131,8)
590 CALL SOUND(1000,349,2,175,8)
600 CALL CHAR(96,"0000000000000000103060F1F0F0606000000000000000E0F0F8F8F8F060
60")
610 CALL CHAR(100,"0000000000000000070F1F1F1F0F0606000000000000000080C060F0F8F06
060")
620 CALL CHAR(112,"0103030301010303030303030101000180E040E0C080C0E0E0E0E0E0C0C08
0E")
630 CALL CHAR(116,"0103030301010F1F373303070E1C1C0E80E040E0C68EFCF0E0E0E0E070361
C18")
640 CALL CHAR(120,"0103030301011F3F3303377F5E00000080E040E6C68CBCF8F0E0F93F0E")
650 CALL CHAR(124,"0107020703010307070707070303010780C0C0C08080C0C0C0C0C0C08")
660 CALL CHAR(128,"0107020763713F0F07070707 0E6C381880C0C0C08080F0F8ECCCC0E070383
870")
670 CALL CHAR(132,"0107026763313B1F0F0F9FFC7000000080C0C0C08080F8FECCC0ECFE7A000
000")
680 CALL CLEAR
690 FOR I=1 TO 9 :: CALL MOTION(#I,0,-20):: NEXT I
700 FOR I=1 TO 8 :: CALL COLOR(I,1,1):: NEXT I
710 RUN "CS1"
```

★★★★

TEIL 2

```
680 ! ***** MINER 99ER *****
690 ! ***** WRITTEN BY *****
700 ! ** H. STAMERJOHANNS **
710 ! **** JAN-DEZ 1984 ****
720 CALL DELSPRITE(ALL):: CALL CLEAR
730 CALL CHAR(136,"0103030319333F1F070703030307070280D8D8D0B0E0E0E0F0F0E0706")
740 CALL SCREEN(2):: CALL MAGNIFY(3)
750 CALL COLOR(1,7,1,2,11,1,3,16,1,4,16,1,5,11,1,6,11,1,7,8,1,10,7,1)
760 ON ERROR 1340
770 DISPLAY AT(1,1)SIZE(3):";<" :: DISPLAY AT(1,20)SIZE(4):"=>" :: DISPLAY AT(1
)4)SIZE(5):USING "#####":SC
780 FOR BI=1 TO 6 :: GOSUB 2160 :: BO=1000 :: FOR I=1 TO 16
790 CALL HCHAR(24,3,38,28):: CALL KEY(1,A,B):: IF B THEN J=1 :: GOTO 820 ELSE CA
LL KEY(2,A,B):: IF B THEN J=2 :: GOTO 820
800 FOR W=1 TO 50 :: NEXT W :: DISPLAY AT(24,8)SIZE(13):"!""#$%hiJklmno" :: GOSU
P 1870 :: FOR W=1 TO 50 :: NEXT W
810 NEXT I :: NEXT BI :: GOTO 770
820 BI=1 :: LI=3 :: SC=0 :: GOTO 880
830 ON ERROR 1340
840 CALL CHARPAT(140,BI$):: BI=VAL(BI$)
850 CALL CHARPAT(141,LI$):: LI=VAL(LI$)
860 CALL CHARPAT(142,SC$):: SC=VAL(SC$)
870 CALL CHARPAT(143,J$):: J=VAL(J$)
880 CALL CHAR(140,"011B1B0B0D0707070F0F070E060000008 0C0C0C098CCFCF8E0E0C0C0C0E0E
040")
890 MU.TM=0 :: GOSUB 2160
900 IF BI=1 OR BI=4 THEN BO=500
910 IF BI=2 OR BI=5 THEN BO=400
920 IF BI=3 THEN BO=700
930 IF BI=6 THEN BO=600
940 FOR I=1 TO LI :: CALL SPRITE(#I+20,112,16,170,13+I*16):: NEXT I :: FOR W=1 T
O 400 :: NEXT W :: CALL DELSPRITE(#21,#22,#23)
950 PA=170 :: PB=29 :: PP=112 :: CALL SPRITE(#3,112,16,170,29)
960 GOSUB 1870 :: K=K-1 :: IF K=0 THEN CALL COLOR(#11,5,#12,3,#13,5,#14,3)
```

```
970 CALL JOYST(J,X,Y):: CALL KEY(J,A,B):: IF B THEN GOSUB 1150 :: GOTO 960
980 IF X THEN CALL GCHAR(PA/8+3,PB/8+1+X/4,H):: GOTO 1060
990 IF Y THEN CALL GCHAR(PA/8+3-Y/2,PB/8+1,H)ELSE 1040
1000 IF H=65 THEN 1090
1010 IF H=80 THEN 1260
1020 IF H=86 THEN 960
1030 IF H>80 AND Y=4 THEN 1640 :: GOTO 960
1040 FOR W=1 TO 150 :: NEXT W
1050 GOTO 960
1060 PP=112-12*(X<0):: CALL PATTERN(#3,PP+4):: CALL LOCATE(#3,PA,PB+X):: PB=PB+2
*X :: CALL SOUND(20,-5,5):: CALL LOCATE(#3,PA,PB):: CALL PATTERN(#3,PP)
1070 ON H-37 GOTO 960,1080,960,1080,1470,1470,960,1080
1080 CALL HCHAR(PA/8+3,PB/8+1,H-1):: EN=EN+1 :: GOTO 960
1090 BO=BO-9 :: FOR I=1 TO 8
1100 CALL PATTERN(#3,136):: CALL SOUND(1,988,5):: PA=PA-Y/2 :: CALL LOCATE(#3,PA
,PB):: CALL PATTERN(#3,140)
1110 CALL SOUND(1,880,4):: PA=PA-Y/2 :: CALL LOCATE(#3,PA,PB):: NEXT I
1120 CALL PATTERN(#3,PP):: CALL SOUND(200,784,7):: CALL GCHAR(PA/8+3,PB/8+1,H)
1130 IF H/2<>INT(H/2)THEN CALL HCHAR(PA/8+3,PB/8+1,H-1):: EN=EN+1
1140 GOTO 960
1150 CALL SOUND(10,-5,3):: PP=120-12*(X<0):: CALL PATTERN(#3,PP)
1160 CALL LOCATE(#3,PA-3,PB+1*X*1.5):: CALL COINC(ALL,C1):: CALL LOCATE(#3,PA-6,
PB+3*X):: CALL COINC(ALL,C2):: CALL GCHAR(PA/8+1,(PB+4*X)/8+1,H)
1170 CALL LOCATE(#3,PA-9,PB+3*X*1.5):: IF H=86 THEN K=7 :: SC=SC+10 :: CALL COLO
P(#11,15,#12,15,#13,15,#14,15)
1180 CALL LOCATE(#3,PA-6,PB+6*X):: CALL COINC(ALL,C3)
1190 CALL LOCATE(#3,PA-3,PB+5*X*1.5):: CALL COINC(ALL,C4):: IF H=86 THEN CALL HC
HAR(PA/8+1,(PB+4*X)/8+1,87)
1200 PB=PB+8*X :: CALL LOCATE(#3,PA,PB):: CALL COINC(ALL,C5)
1210 PP=PP-8 :: CALL PATTERN(#3,PP)
1220 IF C1+C2+C3 AND K=7 THEN K=0 :: GOSUB 1790 ELSE IF C1+C2+C3+C4+C5 THEN GOSU
B 1790
1230 CALL GCHAR(PA/8+3,PB/8+1,H):: ON H-37 GOTO 1250,1240,1250,1240,1470,1470,12
50,1240
1240 CALL HCHAR(PA/8+3,PB/8+1,H-1):: CALL SOUND(10,-1,3):: EN=EN+1
1250 GOTO 960
1260 IF PA<30 AND Y=4 THEN 960
1270 IF PA=10 AND Y=-4 THEN PA=PA+32 :: GOTO 1310
1280 IF PA=42 AND Y=4 THEN PA=PA-32 :: GOTO 1310
1290 IF PA=42 AND Y=-4 THEN PA=PA+64 :: GOTO 1310
1300 IF PA>100 THEN PA=PA-16*Y :: GOTO 1310
1310 CALL SOUND(200,659,8,698,8):: CALL LOCATE(#3,PA,PB)
1320 CALL GCHAR(PA/8+3,PB/8+1,H):: IF H=39 THEN CALL HCHAR(PA/8+3,PB/8+1,38):: E
N=EN+1
1330 GOTO 960
1340 ON ERROR 1340
1350 IF PA/8+3-Y/2>26 THEN RETURN 960
1360 IF PB<38 THEN PB=5 :: PP=124 ELSE IF PB>204 THEN PB=237
1370 CALL PATTERN(#3,PP+8):: IF PA/8+1>20 THEN 1410
1380 CALL MOTION(#3,7,0):: CALL GCHAR(PA/8+7+(BI=3)+(BI=6),PB/8+1,H):: IF H<>32
AND H<>80 THEN 1430
1390 IF H=80 THEN CALL SPRITE(#1,88,6,138,181,#2,92,6,122,181):: CALL DELSPRITE(
#4,#5):: GOTO 1670
1400 CALL POSITION(#3,PA,PB):: IF PA>168 THEN CALL MOTION(#3,0,0)ELSE 1400
1410 FOR I=1 TO 10 :: CALL COLOR(#3,I):: CALL SOUND(-100,-6,10-I,110,2):: NEXT I
1420 GOTO 2110
1430 CALL POSITION(#3,ZA,PB):: IF ZA>PA+28 THEN CALL MOTION(#3,0,0)ELSE 1430
1440 PA=PA+32 :: CALL LOCATE(#3,PA,PB):: CALL PATTERN(#3,PP)
1450 IF H/2<>INT(H/2)THEN CALL HCHAR(PA/8+3,PB/8+1,H-1):: EN=EN+1
1460 ON ERROR 1340 :: RETURN 960
1470 CALL GCHAR(PA/8+7,PB/8+1,H):: IF H=32 THEN 1560 ELSE IF H=40 THEN 1590
```

```
1480 IF BI=1 OR BI=4 THEN IF PA<15 THEN EN2=1
1490 IF BI=2 OR BI=5 THEN IF PA<45 AND PB<50 THEN EN3=1 ELSE IF PA<45 AND PB>190
 THEN EN4=1
1500 CALL HCHAR(PA/8+3,PB/8+1,42):: CALL PATTERN(#3,PP+8):: CALL MOTION(#3,7,0)
1510 CALL POSITION(#3,ZA,PB):: IF ZA>PA+30 THEN CALL MOTION(#3,0,0)ELSE 1510
1520 PA=PA+32 :: CALL LOCATE(#3,PA,PB)
1530 CALL GCHAR(PA/8+3,PB/8+1,H):: IF H>41 THEN 1500 ELSE CALL PATTERN(#3,PP)
1540 IF H=39 THEN CALL HCHAR(PA/8+3,PB/8+1,38):: EN=EN+1
1550 GOTO 960
1560 EN1=1 :: CALL HCHAR(PA/8+3,PB/8+1,42):: CALL PATTERN(#3,132):: CALL MOTION(
#3,7,-7)
1570 CALL POSITION(#3,ZA,PB):: IF ZA>PA+66 THEN CALL MOTION(#3,0,0)ELSE 1570
1580 PA=PA+68 :: CALL LOCATE(#3,PA,PB):: R1=1 :: R2=4
1590 CALL HCHAR(PA/8+3-R1,PB/8+1,42):: CALL PATTERN(#3,120):: CALL MOTION(#3,7,7
)
1600 CALL POSITION(#3,ZA,PB):: IF ZA>PA+30-R2 THEN CALL MOTION(#3,0,0)ELSE 1600
1610 PA=PA+32-R2 :: PB=141 :: CALL LOCATE(#3,PA,PB):: CALL GCHAR(PA/8+3,PB/8+1,H
):: CALL PATTERN(#3,112)
1620 IF H=39 THEN CALL HCHAR(PA/8+3,PB/8+1,38):: EN=EN+1
1630 R1,R2=0 :: GOTO 960
1640 IF H>83 THEN 1970 ELSE CALL SOUND(10,1200,3):: CALL HCHAR(PA/8+1,PB/8+1,32)
:: MU=MU+H-80
1650 CALL POSITION(#4,ZA,ZB):: IF ZB<181 THEN ZB=ZB+8 :: CALL LOCATE(#4,138,ZB,#
5,122,ZB):: GOTO 1650
1660 CALL HCHAR(19,24,80):: GOTO 1050
1670 CALL PATTERN(#3,124):: PP=124
1680 CALL POSITION(#3,PA,PB):: IF PA>122 THEN CALL MOTION(#3,0,0):: CALL COLOR(#
3,1):: CALL HCHAR(19,24,32)ELSE 1680
1690 FOR W=1 TO 50 :: NEXT W
1700 GOSUB 1880 :: CALL JOYST(J,X,Y):: CALL KEY(J,A,B):: IF B THEN 1730
1710 IF X THEN PB=PB+2*X :: CALL LOCATE(#1,138,PB,#2,122,PB,#3,122,PB)ELSE 1690
1720 IF PB>181 OR PB<61 THEN CALL COLOR(#3,16):: GOTO 1340 ELSE 1700
1730 CALL SOUND(100,-6,2):: CALL COLOR(#3,16):: IF MU>3 OR MU<0 THEN 1340 ELSE C
ALL MOTION(#3,-7,0):: PA=106-MU*32 :: CALL GCHAR(PA/8+3,PB/8+1,H)
1740 FOR WAIT=1 TO 100 :: NEXT WAIT :: CALL SPRITE(#4,88,6,138,PB,#5,92,6,122,PB
):: CALL DELSPRITE(#1,#2):: MU=0
1750 GOSUB 1880 :: FOR W=1 TO 10
1760 CALL POSITION(#3,ZA,PB):: IF ZA<PA THEN CALL MOTION(#3,0,0):: CALL LOCATE(#
3,PA,PB):: GOTO 1780 ELSE 1770
1770 NEXT W :: GOTO 1750
1780 IF H=32 THEN 1340 ELSE CALL HCHAR(PA/8+3,PB/8+1,38):: ON ERROR 1340 :: RETU
RN 960
1790 CALL COINC(#4,#3,48,C1):: CALL COINC(#2,#3,48,C2):: IF C1+C2 THEN RETURN EL
SE IF K<1 THEN 1410
1800 CALL COINC(#3,#11,32,C):: IF C THEN I=1 :: GOTO 1850
1810 CALL COINC(#3,#12,32,C):: IF C THEN I=2 :: GOTO 1850
1820 CALL COINC(#3,#13,32,C):: IF C THEN I=3 :: GOTO 1850
1830 CALL COINC(#3,#14,32,C):: IF C THEN I=4 :: GOTO 1850
1840 GOTO 1230
1850 CALL LOCATE(#I+10,I*10+175,100):: MO(I)=I*10+175 :: CALL SOUND(10,988,3)
1860 TM=TM+1 :: GOTO 1230
1870 CALL COINC(ALL,C):: IF C THEN GOSUB 1790
1880 IF MP=96 THEN M=M+4 ELSE M=M-4
1890 CALL LOCATE(#11,MO(1),M1-M,#12,MO(2),M2-M,#13,MO(3),M3-M,#14,MO(4),M4-M)
1900 IF (BI=3)+(BI=6)THEN CALL LOCATE(#15,MO(5),M5-M,#16,MO(6),M6-M)
1910 CALL COINC(ALL,C):: IF C THEN GOSUB 1790
1920 IF M=0 THEN MP=96 ELSE IF M=32+((BI=3)OR(BI=6))*-8 THEN MP=100 ELSE 1940
1930 CALL PATTERN(#11,MP,#12,MP,#13,MP,#14,MP,#15,MP,#16,MP)
1940 BO=BO-1 :: IF BO<0 THEN CALL SOUND(2000,110,2):: GOTO 1340
1950 DISPLAY AT(1,23)SIZE(3):USING "###":BO
1960 RETURN
```

```
1970 ON BI GOTO 1980,1990,2000,1980,1990,2000
1980 IF EN=98 AND EN1=1 AND EN2=1 THEN 2010 ELSE 960
1990 IF EN=97 AND EN3=1 AND EN4=1 THEN 2010 ELSE 960
2000 IF EN=80 THEN 2010 ELSE 960
2010 BI=BI+1 :: IF BI=7 THEN BI=4
2020 SC=SC+BO
2030 RESTORE 3040 :: FOR I=1 TO 6 :: READ S1,S2
2040 CALL SOUND(S1,S2,2):: NEXT I
2050 CALL DELSPRITE(#3):: SC=SC+TM*25+EN :: DISPLAY AT(1,4)SIZE(5):USING "#####"
:SC
2060 BI$=RPT$("0",15)&STR$(BI):: CALL CHAR(140,BI$)
2070 LI$=RPT$("0",15)&STR$(LI):: CALL CHAR(141,LI$)
2080 LSC$=STR$(SC):: SC$=RPT$("0",16-LEN(LSC$))&LSC$ :: CALL CHAR(142,SC$)
2090 J$=RPT$("0",15)&STR$(J):: CALL CHAR(143,J$)
2100 RUN 830
2110 LI=LI-1
2120 RESTORE 3050 :: FOR I=1 TO 11 :: READ S1,S2
2130 CALL SOUND(S1,S2,2):: NEXT I
2140 CALL DELSPRITE(#3):: SC=SC+TM*25+EN :: DISPLAY AT(1,4)SIZE(5):USING "#####"
:SC
2150 IF LI=0 THEN RUN 760 ELSE 890
2160 ON BI GOTO 2220,2230,2240,2220,2230,2240
2170 CALL DELSPRITE(ALL)
2180 FOR I=2 TO 23
2190 READ BI$
2200 DISPLAY AT(I,1)SIZE(29):BI$
2210 NEXT I :: ON BI GOTO 2250,2300,2350,2250,2300,2350
2220 CALL HCHAR(1,15,32,3):: RESTORE 2420 :: GOTO 2170
2230 DISPLAY AT(1,13)SIZE(3):"KJL" :: RESTORE 2620 :: GOTO 2170
2240 CALL HCHAR(1,15,32,3):: RESTORE 2840 :: GOTO 2170
2250 CALL SPRITE(#11,96,5,9,47,#12,96,3,9,207,#13,96,5,41,211,#14,96,3,169,215)
2260 M1=47 :: M2=207 :: M3=211 :: M4=215 :: MO(1)=9 :: MO(2)=9 :: MO(3)=41 :: MO
(4)=169 :: MP=96 :: M=0
2270 TM,EN,EN1,EN2=0
2280 IF BI=1 THEN CALL HCHAR(2,5,86):: CALL HCHAR(6,28,86):: CALL HCHAR(22,27,86
)
2290 RETURN
2300 CALL SPRITE(#11,96,5,105,71,#12,96,3,9,111,#13,96,5,41,227,#14,96,3,169,187
)
2310 M1=71 :: M2=111 :: M3=227 :: M4=187 :: MO(1)=105 :: MO(2)=9 :: MO(3)=41 ::
MO(4)=169 :: MP=96 :: M=0
2320 TM,EN,EN3,EN4=0
2330 IF BI=2 THEN CALL HCHAR(18,28,86):: CALL HCHAR(14,8,86)
2340 RETURN
2350 CALL SPRITE(#4,88,6,138,181,#5,92,6,122,181)
2360 CALL SPRITE(#11,96,5,9,55,#12,96,3,9,227,#13,96,3,41,63,#14,96,5,41,219)
2370 CALL SPRITE(#15,96,5,73,55,#16,96,3,73,227)
2380 M1=55 :: M2=227 :: M3=63 :: M4=219 :: M5=55 :: M6=227
2390 MO(1)=9 :: MO(2)=9 :: MO(3)=41 :: MO(4)=41 :: MO(5)=73 :: MO(6)=73 :: MP=96
 :: M,EN=0
2400 IF BI=3 THEN CALL HCHAR(22,7,83,3):: CALL HCHAR(22,14,82):: CALL HCHAR(22,2
0,81):: CALL HCHAR(22,25,83,3)
2410 RETURN
2420 DATA ,,"' )''''''''')''''''')''''''''+-"
2430 DATA " @        @      @        **"
2440 DATA " A        A      A        **"
2450 DATA " B        B      B        **"
2460 DATA "' )''   ''''    ''''-+''''+-"
2470 DATA "  @              DE      **"
2480 DATA "  A             DE       **"
2490 DATA "  B            DE        **"
2500 DATA "' )'''        DE         **"
```

Halten Sie durch!
Gleich haben Sie's geschafft.
Viel Spaß im „Bergwerk"!

```
2510 DATA " @              DE          **"
2520 DATA " A             DE           **"
2530 DATA " B            DE            **"
2540 DATA "''''''''''  '+-'''''  ' ''''+-"
2550 DATA "       @      FG        @   **"
2560 DATA "       A       FG       A   **"
2570 DATA "       B        FG      B   **"
2580 DATA "''''''''''  ' ''''''''''''''+-"
2590 DATA "             @              **"
2600 DATA "TU           A              **"
2610 DATA "             B              **"
2620 DATA "            HPI"
2630 DATA "            H I"
2640 DATA "''''-+''''''''''''''''''''''+-''''"
2650 DATA "   **       KJL         **"
2660 DATA "   **       HPI         **"
2670 DATA "   **       H I         **"
2680 DATA "''''-+''''  '''''  ''''''''''''"
2690 DATA "   **"
2700 DATA "   **        P"
2710 DATA "   **"
2720 DATA "' ''''''''"
2730 DATA " @           KJL"
2740 DATA " A           HPI"
2750 DATA " B           H I"
2760 DATA "''''''''''  '''''  ' ''''''''''"
2770 DATA "       @           @"
2780 DATA "       A     P     A"
2790 DATA "       B           B"
2800 DATA "'''' '  '''        '''  '''' '"
2810 DATA "  @          KJL          @"
2820 DATA "TUA          HPI          A"
2830 DATA "  B          H I          B"
2840 DATA ' ,"'''+-'''                '''-+'''"
2850 DATA "  **                       **"
2860 DATA "  **                       **"
2870 DATA "  **                       **"
2880 DATA "'''+-''''              ''''-+'''"
2890 DATA "  **                       **"
2900 DATA "  **                       **"
2910 DATA "  **                       **"
2920 DATA "'''+-''''''''        ''''''''-+'''"
2930 DATA "  **                       **"
2940 DATA "  **                       **"
2950 DATA "  **                       **"
2960 DATA "'''' ''                  '' ''''"
2970 DATA "   @                      @"
2980 DATA "   A                      A"
2990 DATA "   B                   P  B"
3000 DATA "'' ''''' ''''' ''''' ''''' ''''' '"
3010 DATA "  @     @     @     @     @ "
3020 DATA "TUA  S  A R R A Q Q A  S  A "
3030 DATA "  B     B     B     B     B "
3040 DATA 190,524,250,659,190,698,190,784,190,659,750,784
3050 DATA 380,175,380,175,190,175,750,175,750,208,190,196
3060 DATA 380,196,190,175,380,175,380,165,750,175
```

Apple Game-Help: Die „große Hilfe"!

Wer hat das nicht schon oft bemerkt und bemängelt: bei vielen professionellen Adventure- oder Action-Spielen sind teilweise die Schwierigkeitsgrade so hoch gewählt, daß nicht nur „Otto Normalverbraucher" sondern auch mancher Profi die Segel streichen muß. Um nicht zuletzt auch dem Anfänger die Möglichkeit zu geben, alle oder zumindest mehrere Level bzw. Bilder durchzuspielen, sind hier für die bekanntesten Spiele ein paar nützliche Hilfen - zum „Manipulieren" - abgedruckt. Zu jedem Spiel ist eine genaue Tastenfolge angegeben, um die Änderung zu aktivieren. Viel Freude an der „Manipulation" (M. K.)

Tips & Tricks

Apple II

Hilfe für „Bolo":

BLOAD BOLO
CALL-151
14A8:EA EA EA
1D38:EA EA EA
1100G

Hilfe für „Bug Battle":

RUN BUG BATTLE
RESET
3FE6:EA EA EA
800G

Hilfe für „Money Munchers":

RUN MONEY MUNCHERS
RESET
1108:EA EA EA EA EA
108A:EA EA
FE7G

Hilfe für „RASTER BLASTER":

BLOAD RASTER BLASTER
CALL-151
8075:EA EA EA
900G

„LIST SOFT" - ein Club stellt sich vor

„LIST SOFT" ist ein Club, der alle jugendlichen Benutzer der Computer TI 99/4A und VC-20 ansprechen soll. Wir wollen versuchen, ungefähr jeden Monat eine Zeitschrift zu entwickeln, an der alle Mitglieder mitwirken sollten. Unsere Clubzentrale ist zwar in Hof, doch sollten Mitglieder aus ganz Deutschland im Club vertreten sein. Der Club besteht derzeit aus zehn „Leuten". Der Clubbeitrag beträgt monatlich drei Mark. Das scheint auf den ersten Blick viel zu sein, aber von diesem Geld wird die Zeitschrift finanziert und jedem Mitglied gratis zugesendet. Wir wollen versuchen, Programme, Tips, Tricks und Programm-Techniken auszutauschen.

Kontaktadressen:
Michael Eckert
Zobelsreutherstr. 30
8670 Hof (Saale)
oder
Ralf Hopperdietzel
Theodor-Storm-Str. 22
8670 Hof (Saale)

Neuer „TI 99"-Club!

In Achim bei Bremen wurde unlängst ein neuer TI 99-Club aus der Taufe gehoben: der „Monstervision Club".
Einer der „Initiatoren", Volker Niemeyer, gab der COMPUTRONIC-Redaktion folgende „Lage-Beschreibung": „Weil sich doch sehr viele vom TI 99/4A abgewandt haben und ihn totschweigen, - sehr zum Nachteil der zahlreichen 99/4A-Besitzer - haben wir uns entschlossen, einen TI 99/4A-Computer-Club zu gründen."
Ein Clubbeitrag wird nicht erhoben. Jeden Monat erscheint eine 32seitige „Fanzine" mit vielen Tests, Infos und natürlich Listings. Wer mehr über den neuen Club erfahren möchte, der wende sich bitte an den

Monstervision Club
Uesener Ring 30
2807 Achim

Ein kostenloses Info-Blatt kommt prompt in's Haus! (M. K.)

ZX-Spectrum – Tips & Tricks

Diese ist das erste Spectrum-Buch, das von der Firma Data Becker herausgegeben wurde.
Es enthält eine ganze Serie von Tips, die teilweise durchaus brauchbar sind. Auch einige Routinen (z. B. 64 Zeichen) und gute Programmierhinweise z. B. zum Thema Dateiverarbeitung sind enthalten. Ebenso sind auffähgige Programme mit recht interessanten Zielen (Grafik, Spielchen u.a.m.) dabei. Übrigens: das Buch ist vollständig auf Tasword II erstellt. Die Druckqualität ist gut, auch die Listings sind ausgezeichnet lesbar. Das Inhaltsverzeichnis ist übersichtlich gegliedert, man findet, was man sucht.
Die Entwicklung der enthaltenen Programme erfolgt schrittweise mit ausführlichen Erläuterungen. So ist es mit zunehmendem Wissen des Einzelnen durchaus möglich, die angebotenen Programme selbst nach eigenen Bedürfnissen zu erweitern oder zu ändern.

Dieses Buch ist für Einsteiger in die Materie interessant.

von:
Peter Bosetti, ISBN 3-89011-075-4

Nebenbei bemerkt . . .

„Siehst Du? Die streiten sich schon wieder um die neue COMPUTRONIC."

„Hippo" und „Birdie" sind bestürzt. Den beiden kann dies nicht passieren - sie haben beide ein Abo!

Gehen auch Sie diesen „Streitigkeiten" aus dem Weg!

Ein COMPUTRONIC-Abo ist lohnenswert!

(Abo-Abrufkarte auf S. 45)

Ein Abenteuer in 3 D!

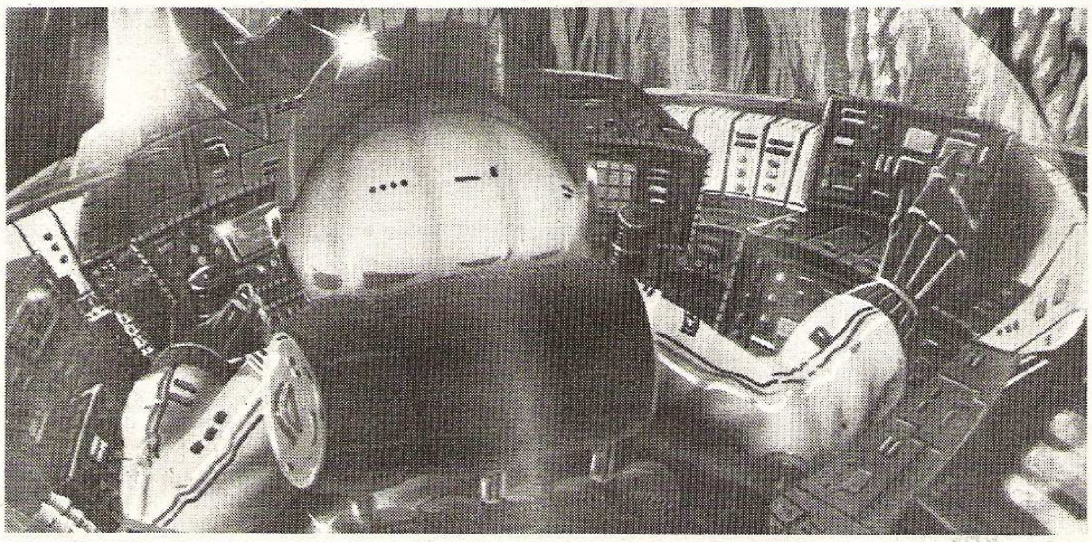
STARDUST

für den TI-99/4A

TEXAS INSTRUMENTS

Ein TOP-Programm, das es in sich hat!

Für alle „Texaner" lautet die Mission:



DELTA 4-A und OMEGA 99 bewachen!

STARDUST ist der Name Ihres Raumschiffes, das Sie als Bordschütze durch die Galaxis steuern. Sie fliegen über die Oberfläche des Planeten OMEGA 99. Achtung: Feindliche Verbände greifen an! Sie haben nun die Aufgabe, mit gezieltem Sperrfeuer diese hinterlistigen – und vor allem schnellen – Burschen am Vorbeiflug zu hindern.

Für diese schwierige Mission stehen Ihnen vier Bomber, Marke „Stardust", zur Verfügung. Pro Spielrunde haben Sie eine bestimmte Anzahl an Gegnern abzuschießen. Und beachten Sie, daß Ihre Energie begrenzt ist! Treffen Sie also schnell, so haben Sie stets die nötige Energie für Ihren Einsatz-Flug.

Wenn es Ihnen gelungen ist, den Planeten Omega 99 erfolgreich zu verteidigen, werden Sie zum Flug in den Raumsektor Delta 4-A „abkommandiert", wo ein weiterer feindlicher Flottenverband versucht, seinen Heimatplaneten zu erreichen. Gelingt es auch nur einem einzigen Feind-Raumschiff, den linken Bildschirmrand „zu durchbrechen", ist Ihre Mission gescheitert. „Andersherum wird auch ein Schuh 'draus": Gelingt es Ihnen nämlich, den Flottenverband aufzuhalten, so werden sich die Feinde selbst vernichten, und Sie kehren als „Triumphator" zurück zum Stützpunkt! (M. K.)

Steuerung der STARDUST auf „Omega 99": Joystick in Nullstellung = gerader Flug. Plus Feuertaste = gerader Schuß nach vorn. Joystick links-rechts = schräger Flug. Plus Feuertaste = Schuß in der entsprechenden Richtung. (Wenn Sie „aus dem Bild" fliegen, verlieren Sie einen Stardust). Steuerung der „Stardust" im Raumsektor „delta 4-A": Joystick gerade = Nullstellung. Plus Feuertaste = Auslösen einer Abwehr-Rakete. Joystick links-rechts = Flug in dieser Richtung. Plus Feuertaste = Auslösen der Abwehr-Rakete. Achtung! Nach Auslösen der Abwehr-Rakete übernimmt der Joystick das Lenken des Geschosses!

Wichtig!

- Bei der Eingabe des Listings die Zeilen 10 bis 39 nicht abtippen, da sonst der Speicher ohne Erweiterung nicht ausreicht.
- In Zeile 500 ist darauf zu achten, daß die Buchstabenfolge „DFDFDF" bei gleichzeitig gedrückter „CNTR"-Taste eingegeben werden muß. Da es sich um frei definierte Zeichen handelt, werden diese erst nach „RUN" des Spiels sichtbar.
- Ist alles richtig eingegeben, erwartet Sie ein Spiel mit tollem 3D-Effekt mit irrer Farbgraphic!

```
10 ! **********************
11 ! *  S T A R D U S T   *
12 ! *  ===============   *
13 ! *                    *
14 ! *    COPYRIGHT BY    *
15 ! *    W.H.DOELTSCH    *
16 ! *                    *
17 ! * 6108 WEITERSTADT 3 *
18 ! * ------------------ *
19 ! * A.D.HINTERSTEIN 10 *
20 ! *  TEL.06150-40547   *
21 ! **********************
22 ! * BENOETIGTE GERAETE *
23 ! * TI 99/4A   KONSOLE *
24 ! *  EXTENDED - BASIC  *
25 ! *      RECORDER      *
26 ! *                    *
27 ! *  SPEICHERBELEGUNG  *
28 ! * NACH RUN UND BREAK *
29 ! * CA.300 BYTES FREE! *
30 ! **********************
31 ! *   A C H T U N G    *
32 ! *                    *
33 ! * DIE ZEILEN  1 - 39 *
34 ! * SIND NUR INFORMATO-*
35 ! * RISCH. SIE DUERFEN *
36 ! * AUS SPEICHERPLATZ- *
37 ! * GRUENDEN NICHT EIN-*
38 ! * GEGEBEN WERDEN.    *
39 ! **********************
100 CALL CLEAR :: CALL SCREEN(2):: LEV=1 :: CALL MAGNIFY(3):: RANDOMIZE :: DIM B
$(7)
110 CALL COLOR(3,16,1,4,16,1,9,16,1,10,16,1,11,16,1,12,16,1,13,6,1):: GOTO 620
120 DATA 40,0000010387CFDFBF,41,80C0E0E1F3FFFBFD,42,00010387CDEDDDBE,43,C0E0F0F8
DC5E5F3E
130 DATA 44,0000041E337575F3,45,000001071F7FBFDF,46,48DCEEF7FBFDFEFF
140 DATA 64,0,65,0028542854101038,66,0000000000020502,67,05010103,68,00000000008
0408
150 DATA 69,4000008,70,54AA54AA541038FE,71,0000000005 0A050A,72,0501030F,73,00000
00040A040A
160 DATA 74,400080E,75,AAAAAAAAAAAAAAAA,35,0000387CAA7C,47,00000001,58,8,59,0000
0000001,33,FFFFFFFFFFFFFFFF
170 DATA 120,000201070102,121,0,122,004080E0804,123,0,128,845A773D0CBF7715,129,0
E9F770E0B0C102,130,06193053B6FEFEDC
180 DATA 132,0D000119E7271B,133,0,134,B0008098E7E4D8,135,0,136,0106182201030F13,
137,337C8,138,0001067CC8C8F0C,139,8
190 DATA 140,0080603E13130F03,141,01,142,8060184480C0F0C8,143,CC3E01,36,0,37,010
3070F1F7FFF0C,38,0,39,80C0E0F0F8FEFF3
200 DATA 60,0101010305050001,61,0002000500020001,62,00000080404,63,00800040008,1
31,E9FAFE4C38CC4244
210 DATA 76,0619F60F,77,0,78,0080F,79,0
220 DATA 76,004020150E1F0E15,77,204,78,00408,79,804
230 DATA 76,030E3CE71C0E,77,0,78,8008C290C4,79,0
240 DATA 76,183CE75A3C18,77,0,78,0,79,0
250 DATA 76,BDDB3C42,77,0,78,0,79,0
260 DATA 76,031CFF1C03,77,0,78,0,79,0
270 DATA 76,01030E0301,77,0,78,80C070C08,79,0,80,0103071D070301,81,0,82,80C0E0B8
E0C08,83,0
280 DATA 84,01030F390F0301,85,0,86,80C0F09CF0C08,87,0,88,01071F791F0701,89,0,90,
80E0F89EF8E08,91,0
290 DATA 92,01030F3FF93F0F03,93,01,94,80C0F0FC9FFCF0C,95,8
300 DATA 76,001866FF3C,77,0,78,0,79,0,80,001E2DF37F1E,81,0,82,000000C08,83,0
```

Und: Achten Sie auf „Buggin' Buster", den Fehler-Teufel! Geben Sie Ihm keine Chance bei der Eingabe des Programms!

```
310 DATA 84,000F3FEFF07F1F,85,0,86,0000C070F0E08,87,0,88,00070F7FF7D87F3B,89,0F,
90,0080C0F8BC6CF870,91,C
320 DATA 92,03073F77FBDCFF3B,93,0F,94,C0E0FCEEDF3BFEDC,95,F
330 DATA 76,0000000000000103,77,02,78,0000000000000080C,79,4,80,0000000000030607,8
1,0606
340 DATA 82,0000000000C060E,83,606
350 DATA 84,0000000000071D16,85,17141C,86,000000000E0B868,87,E82838,88,00000000
0F393E27
360 DATA 89,2C2C243C,90,00000000F09C7CE4,91,3434243C
370 DATA 92,0000000F39393E27,93,2C2C2C2C243C,94,000000F09C9C7CE4,95,34343434243C
380 DATA 76,0001030D0301,77,0,78,0080C070C08,79,0,80,0003071A0703,81,0,82,00C0E0
B8E0C,83,0
390 DATA 84,00030F35350F03,85,0,86,00C0F05C5CF0C,87,0,88,00030F75750F03,89,0,90,
00C0F05E5EF0C,91,0
400 DATA 92,00071FEAEA1F07,93,0,94,00E0F8AFAFF8E,95,0
410 B$(1)="@A@@@A@@@A@@@AA@@@A@@@A@@@A"&RPT$("@",32)&"A@@@A@@@A@@@@A@@@A@@@A"&RP
T$("@",31)
420 B$(2)="@F@@@F@@@F@@@@@@@@F@@@F@@@F"&RPT$("@",32)&"F@@@F@@@@@@@@@@@@F@@@F"&RP
T$("@",31)!! B$(1)=B$(1)&B$(2)
430 B$(2)="BD@@BD@@BD@@BDBD@@BD@@BD@@BDCE@@CE@@CE@@CECE@@CE@@CE@@CE@@BD@@BD@@BD@
@@@BD@@BD@@BD@@@@CE@@CE@@CE@@@@CE@@CE@@CE@@"
440 B$(3)="GI@@GI@@GI@@@@@@@@GI@@GI@@GIHJ@@HJ@@HJ@@@@@@@@HJ@@HJ@@HJ@@GI@@GI@@@@@
@@@@@@@GI@@GI@@@@HJ@@HJ@@@@@@@@@@@@@HJ@@HJ@@" !! B$(2)=B$(2)&B$(3)
450 B$(3)=RPT$("@",28)&"A@@@A@@@A@@@A@@@A@@@A@@@A@@@A"&RPT$("@",30)&"A@@@A@@@A@@@
@@@A@@@A@@@A@@"
460 B$(4)=RPT$("@",28)&"F@@@F@@@F@@@@@@@@@@@F@@@F@@@F"&RPT$("@",30)&"F@@@F@@@@@@@
@@@@@@@F@@@F@@" !! B$(3)=B$(3)&B$(4)
470 B$(4)="@CE@@CE@@CE@@@@@@CE@@CE@@CE@D@@BD@@BD@@BD@@BD@@BD@@BD@@BE@@CE@@CE@@CE
@@CE@@CE@@CE@@C@BD@@BD@@BD@@@@@@BD@@BD@@BD@"
480 B$(5)="@CE@@CE@@CE@@@@@@CE@@CE@@CE@I@@GI@@GI@@@@@@@@@@GI@@GI@@GJ@@HJ@@HJ@@@@
@@@@@@HJ@@HJ@@H@GI@@GI@@@@@@@@@@@@@@GI@@GI@" !! B$(4)=B$(4)&B$(5)
490 B$(5)=": /: ; /  :/ ;  / ; / : /:/; score 000000 : fuel left /:; / ; / / ; /
:KKKKKKKKKKKK "
500 B$(6)=" DFDFDF /; /: ;/: : ; :/  ; / ; ;/ :; /: / : / ; / ;/ :  ; / ; / : /
; : / / :/ : ;; / / ; / ;/ : / : / ;  ;/ ; "
510 B$(7)=":; : /; /: : ;; / : /: ;/ ; ; :/ :  / :  ; : / ; /; :; / : ; / ;: / :
 ; ; / ;/: /;/;;: /  ; / ; :/ ; ; /: ; /: /" !! RETURN
520 DATA 9,6,3,10,1,9,1,2,6,2,5,13,12,6,13,12
530 DATA 14,5,3,12,1,14,1,2,6,2,5,13,12,6,13,12
540 DATA 13,4,4,14,1,13,1,2,14,2,5,9,15,6,9,15
550 DATA 14,4,4,14,1,14,1,2,5,2,5,13,12,6,13,12
560 DATA 7,3,5,16,1,7,1,2,11,2,5,14,8,6,14,8
570 DATA 5,3,5,16,1,5,1,2,8,2,5,7,15,6,7,15
580 DATA 5,2,6,18,1,5,1,2,11,2,5,14,8,6,14,8
590 DATA 13,2,6,18,1,13,1,2,7,2,5,14,12,6,14,12
600 DATA 7,1,7,20,1,7,1,2,14,2,5,9,15,6,9,15
610 DATA 14,1,7,20,1,14,1,2,6,2,5,13,12,6,13,12
620 CALL COLOR(1,14,1,2,6,2,5,13,12,6,13,12)!! FOR I=1 TO 52 !! READ A,C$ !! CAL
L CHAR(A,C$)!! NEXT I !! GOSUB 410 !! AN=10
630 CALL HCHAR(23,1,33,128)!! CALL VCHAR(1,31,33,96)
640 DISPLAY AT(14,1):RPT$("( )*+,-.",4)
650 DISPLAY AT(3,1):B$(5)&B$(6):B$(7)!! DISPLAY AT(15,1):B$(1)!! DISPLAY AT(13,5
)SIZE(19):"Press fire to begin"
660 DISPLAY AT(2,7)SIZE(15):"s t a r d u s t" !! CALL SPRITE(#1,132,5,157,120,#2
,36,2,159,120)!! DISPLAY AT(24,5)SIZE(20):"c1985 willi doeltsch"
670 CALL POSITION(#1,Y,X)!! GOSUB 1580 !! SP=X-95 !! ZE=Y-125 !! GOSUB 1580 !! C
ALL KEY(1,T,S)!! IF T=18 THEN 800
680 A=-14*SGN(SP)+RND*7 !! GOSUB 1580 !! FA=-2*SGN(ZE)+RND*2 !! GOSUB 1580 !! CA
LL SOUND(-1000,110,30,800,30,850+(X*2),30,-8,10)
690 CALL MOTION(#1,FA,A,#2,FA,A)!! GOSUB 1580 !! CALL KEY(1,T,S)!! IF T<>18 THEN
 670 ELSE 800
700 RESTORE 300 !! GOSUB 1540 !! RESTORE 520 !! GOSUB 1550 !! RETURN
```

```
710 RESTORE 270 :: GOSUB 1540 :: RESTORE 530 :: GOSUB 1550 :: RETURN
720 RESTORE 330 :: GOSUB 1540 :: RESTORE 540 :: GOSUB 1550 :: RETURN
730 RESTORE 380 :: GOSUB 1540 :: RESTORE 550 :: GOSUB 1550 :: RETURN
740 RESTORE 300 :: GOSUB 1540 :: RESTORE 560 :: GOSUB 1550 :: RETURN
750 RESTORE 270 :: GOSUB 1540 :: RESTORE 570 :: GOSUB 1550 :: RETURN
760 RESTORE 330 :: GOSUB 1540 :: RESTORE 580 :: GOSUB 1550 :: RETURN
770 RESTORE 380 :: GOSUB 1540 :: RESTORE 590 :: GOSUB 1550 :: RETURN
780 RESTORE 300 :: GOSUB 1540 :: RESTORE 600 :: GOSUB 1550 :: RETURN
790 RESTORE 270 :: GOSUB 1540 :: RESTORE 610 :: GOSUB 1550 :: RETURN
800 CALL DELSPRITE(ALL):: DISPLAY AT(13,5)SIZE(19):": /;; : /;/ ; : /:;" :: GOSU
B 1580
810 DISPLAY AT(24,5)SIZE(20):"Prepare for level";LEV :: ON LEV GOSUB 700,710,720
,730,740,750,760,770,780,790
820 CALL SOUND(200,600,0,700,0,-2,0):: LA=224 :: CALL HCHAR(6,30-H,35,H):: CALL
HCHAR(24,5,33,23):: HH=H
830 CALL SPRITE(#1,132,5,157,120,#2,36,2,159,120,#23,36,2,27,LA)
840 LA=LA-1 :: IF LA<123 THEN GOSUB 1050 :: GOTO 830 :: ELSE CALL LOCATE(#23,27,
LA):: IF H=0 THEN 1090
850 CALL MOTION(#1,0,0,#2,0,0):: CALL SPRITE(#4,76,FA,104,INT(RND*(176-64))+64,S
PE,SPE*SGN(RND-.5)):: GOSUB 1580
860 FOR I=80 TO 92 STEP 4 :: CALL PATTERN(#4,I):: CALL SOUND(-2350,110,30,800,30
,1000,30,-8,18):: FOR A=1 TO LP
870 CALL JOYST(1,X,Y):: IF X=4 THEN CALL PATTERN(#1,140):: GOTO 910 ELSE IF X=-4
 THEN CALL PATTERN(#1,136):: GOTO 940 ELSE IF FL THEN 890
880 CALL MOTION(#1,0,0,#2,0,0):: CALL PATTERN(#1,132):: FL=-1 :: CALL POSITION(#
1,ZE,SP):: IF (SP<20)+(SP>228)THEN GOSUB 1050 :: GOTO 830
890 CALL KEY(1,T,S):: IF T<>18 THEN 970 ELSE CALL POSITION(#1,ZE,SP)
900 CALL SPRITE(#3,120,7,ZE,SP,-17,0):: GOSUB 1560 :: IF HIT THEN 840 ELSE 980
910 CALL POSITION(#1,ZE,SP):: IF SP>220 THEN GOSUB 1050 :: GOTO 830 ELSE FL=0
920 CALL MOTION(#1,0,18,#2,0,18):: CALL KEY(1,T,S):: IF T<>18 THEN 970 ELSE CALL
 MOTION(#1,0,0,#2,0,0)
930 CALL POSITION(#1,ZE,SP):: CALL SPRITE(#3,120,7,ZE-5,SP+4,-17,17):: GOSUB 156
0 :: IF HIT THEN CALL PATTERN(#1,132):: FL=0 :: GOTO 840 ELSE 980
940 CALL POSITION(#1,ZE,SP):: IF SP<32 THEN GOSUB 1050 :: GOTO 830 ELSE FL=0
950 CALL MOTION(#1,0,-18,#2,0,-18):: CALL KEY(1,T,S):: IF T<>18 THEN 970 ELSE CA
LL MOTION(#1,0,0,#2,0,0)
960 CALL POSITION(#1,ZE,SP):: CALL SPRITE(#3,120,7,ZE-5,SP-6,-17,-17):: GOSUB 15
60 :: IF HIT THEN CALL PATTERN(#1,132):: FL=0 :: GOTO 840 ELSE 980
970 GOSUB 1580 :: NEXT A :: LA=LA-1 :: CALL LOCATE(#23,27,LA)
980 GOSUB 1580 :: NEXT I :: CALL DELSPRITE(#4):: LA=LA-1 :: CALL LOCATE(#23,27,L
A):: GOTO 840
990 CALL DELSPRITE(#4):: CALL HCHAR(5,19,75,11):: GOTO 810
1000 CALL DELSPRITE(#3):: CALL PATTERN(#4,128):: CALL SOUND(-400,-7,0):: CALL MO
TION(#4,3,3):: CALL SOUND(-400,-6,6):: CALL PATTERN(#1,132)
1010 GOSUB 1580 :: CALL POSITION(#4,ZE,SP):: CALL SPRITE(#5,128,7,ZE,SP,3,-3)::
GOSUB 1580 :: CALL POSITION(#1,ZE,SP):: CALL LOCATE(#2,159,SP)
1020 CALL SOUND(-400,-6,12):: GOSUB 1580 :: CALL SOUND(-600,-6,16):: CALL DELSPR
ITE(#4,#5)
1030 GOSUB 1580 :: CALL HCHAR(6,30-H,32):: H=H-1
1040 SCORE=SCORE+(LEV*50):: DISPLAY AT(4,7)SIZE(7):SCORE :: RETURN
1050 CALL MOTION(#1,-1,1,#2,-1,-1):: CALL SOUND(-400,-7,0):: CALL PATTERN(#1,128
,#2,128)
1060 CALL HCHAR(6,AN,32,2):: AN=AN-2 :: IF AN=2 THEN 1490 ELSE CALL HCHAR(6,AN,3
2,2):: CALL DELSPRITE(ALL)
1070 LA=224 :: CALL LOCATE(#23,27,LA):: CALL HCHAR(6,30-HH,35,HH):: H=HH
1080 RETURN
1090 CALL DELSPRITE(ALL):: DISPLAY AT(11,1):B$(7)&B$(7):B$(7)
1100 DISPLAY AT(24,3)SIZE(24):"Prepare for special game"
1110 RESTORE 200 :: FOR I=1 TO 16 :: READ A,C$ :: CALL CHAR(A,C$):: NEXT I
1120 A=64 :: FOR I=17 TO 22 :: CALL SPRITE(#I,132,1,A,245):: A=A+16 :: NEXT I ::
 A=64 :: FOR I=23 TO 28 :: CALL SPRITE(#I,132,1,A,8):: A=A+16 :: NEXT I
```

```
1130 ON LEV GOTO 1190,1150,1160,1170,1180,1190,1200,1190,1150,1160
1140 RESTORE 210 :: GOTO 1210
1150 RESTORE 300 :: GOTO 1210
1160 RESTORE 230 :: GOTO 1210
1170 RESTORE 270 :: GOTO 1210
1180 RESTORE 250 :: GOTO 1210
1190 RESTORE 240 :: GOTO 1210
1200 RESTORE 220
1210 FOR I=1 TO 4 :: READ A,C$ :: CALL CHAR(A,C$):: NEXT I
1220 A=64 :: FOR I=4 TO 9 :: CALL SPRITE(#I,76,INT(RND*13)+3,A,230,0,INT(RND*-3)
+-1):: A=A+16 :: NEXT I :: CALL HCHAR(24,3,33,26)
1230 CALL SPRITE(#1,132,5,163,120):: CALL SOUND(100,1400,0)
1240 FOR I=1 TO 10+(LEV*3)
1250 CALL COINC(ALL,HIT):: IF HIT THEN 1430
1260 CALL JOYST(1,X,Y):: IF X=4 THEN 1290 ELSE IF X=-4 THEN 1270 ELSE CALL MOTIO
N(#1,0,0):: GOTO 1310
1270 CALL COINC(ALL,HIT):: IF HIT THEN 1430
1280 CALL POSITION(#1,ZE,SP):: IF SP<32 THEN CALL MOTION(#1,0,0):: GOTO 1310 ELS
E CALL MOTION(#1,0,-20):: GOTO 1310
1290 CALL COINC(ALL,HIT):: IF HIT THEN 1430
1300 CALL POSITION(#1,ZE,SP):: IF SP>214 THEN CALL MOTION(#1,0,0):: GOTO 1310 EL
SE CALL MOTION(#1,0,20)
1310 CALL KEY(1,T,S):: IF T<>18 THEN 1250 ELSE CALL MOTION(#1,0,0):: CALL POSITI
ON(#1,ZE,SP):: CALL SOUND(-1570,800,30,800,30,2500,30,-8,12)
1320 CALL SPRITE(#2,60,16,ZE-6,SP,-20,0)
1330 FOR H=1 TO 16 :: CALL KEY(1,T,S):: IF T=18 THEN 1350 ELSE CALL JOYST(1,X,Y)
:: CALL MOTION(#2,-20,X)
1340 NEXT H :: CALL DELSPRITE(#2):: CALL DELSPRITE(#2):: CALL COINC(ALL,HIT):: I
F HIT THEN 1370 ELSE 1260
1350 CALL SOUND(-250,-6,8):: CALL POSITION(#2,ZE,SP):: A=INT((ZE/8)/2):: CALL CO
INC(#2,#A,8,HIT):: CALL PATTERN(#2,128)
1360 IF HIT THEN 1370 ELSE CALL DELSPRITE(#2):: CALL DELSPRITE(#2):: GOTO 1250
1370 CALL LOCATE(#A,(A*8)*2,230):: CALL DELSPRITE(#2):: CALL SOUND(300,-7,5):: G
OSUB 1040
1380 NEXT I
1390 CALL MOTION(#1,0,0):: FOR I=9 TO 4 STEP -1 :: CALL SOUND(-280,-7,I*2-7):: C
ALL PATTERN(#I,128):: CALL MOTION(#I,0,0):: FOR A=1 TO 55
1400 NEXT A :: CALL DELSPRITE(#I):: NEXT I :: LEV=LEV+1 :: IF LEV>10 THEN 1490
1410 CALL SOUND(900,195,5,246,5,349,5):: CALL SOUND(4250,261,8,329,8,391,8)
1420 CALL DELSPRITE(ALL):: DISPLAY AT(14,1):RPT$("()*+,-.",4):: DISPLAY AT(15,1)
:B$(1):: GOTO 810
1430 CALL MOTION(#1,0,0):: FOR I=1 TO 5 :: CALL COLOR(1,INT(RND*13)+3,2):: NEXT
I :: CALL COLOR(1,FA,2)
1440 CALL POSITION(#1,A,FA):: FOR I=4 TO 9 :: CALL MOTION(#I,0,0):: CALL POSITIO
N(#I,ZE,SP):: CALL PATTERN(#I,84):: CALL MOTION(#I,(A-ZE)/2,(FA-SP)/2)
1450 CALL COINC(#I,#1,8,HIT):: IF HIT THEN 1460 ELSE 1450
1460 CALL DELSPRITE(#I):: CALL PATTERN(#1,128):: CALL SOUND(100,-7,0):: CALL PAT
TERN(#1,132)
1470 FOR X=1 TO 5 :: CALL COLOR(#1,16,#1,5):: NEXT X :: NEXT I :: CALL PATTERN(#
1,128)
1480 FOR I=2 TO 5 :: CALL SOUND(-900,-7,I):: CALL SPRITE(#I,128,INT(RND*13)+1,A,
FA,INT(RND*-2)+-2,INT(RND*2)+2*SGN(RND-.5)):: NEXT I :: GOSUB 1050 :: GOTO 1420
1490 CALL DELSPRITE(ALL):: IF SCORE<=HIC THEN 1500 ELSE HIC=SCORE
1500 CALL SPRITE(#1,132,5,157,120,#2,36,2,159,120)
1510 DISPLAY AT(8,4)SIZE(22):"highscore        ";HIC :: DISPLAY AT(10,4)SIZE(22):
"spielpunkte      ";SCORE
1520 DISPLAY AT(12,4)SIZE(22):"neues spiel       J  n" :: DISPLAY AT(14,1):RPT$("
()*+,-.",4):: GOSUB 1580
1530 CALL KEY(3,T,S):: IF T=74 THEN SCORE=0 :: AN=10 :: LEV=1 :: GOTO 650 ELSE I
F T=78 THEN CALL CLEAR :: END ELSE GOSUB 1580 :: GOTO 1530
1540 FOR I=1 TO 20 :: READ A,C$ :: CALL CHAR(A,C$):: GOSUB 1580 :: NEXT I :: HH=
```

```
1550 READ FA,LP,SPE,H :: FOR I=1 TO 4 :: READ X,Y,SP :: CALL COLOR(X,Y,SP):: GOS
UB 1580 :: NEXT I :: RETURN
1560 FOR X=1 TO 4 :: CALL COINC(#3,#4,10,HIT):: IF HIT THEN 1000
1570 GOSUB 1580 :: NEXT X :: CALL DELSPRITE(#3):: RETURN
1580 DISPLAY AT(15,1):B$(F):: F=F+1 :: IF F=5 THEN F=1
1590 RETURN
```

✽✽ TOP! | STARDUST | TOP! ✽✽

TOP-PRO-GRAMM!

SPIDER MUTANTS

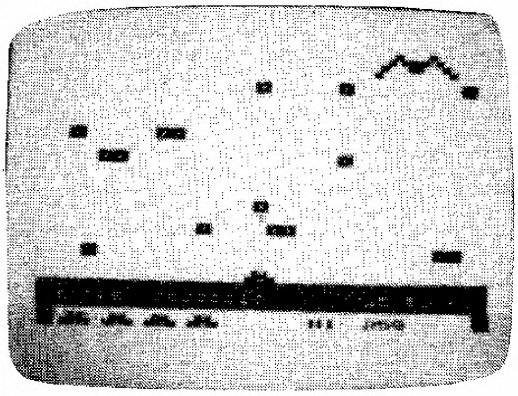
ZX-81 mit 16k RAM

Eine gewaltige Spinne will Ihnen „an's Leder"!

SPIDER MUTANTS ist der Kampf gegen eine garstige Galaxis-Spinne, die versucht, alles Leben auf dem Planeten „Sinclair" zu zerstören! Sie haben nun die schwierige Aufgabe übernommen, mit Ihrem Raumpanzer dieser Kreatur den Garaus zu machen.

Das Spiel SPIDER MUTANTS ist ein Programm für den ZX-81 mit mindestens 16K RAM. Es verfügt über fünf Schwierigkeits-Stufen – Stufe 5 ist nur für „Profis".
Während des Spielablaufs zeigen Kontroll-Instrumente ständig die aktuellen Scores, High-Scores und Levels an; ebenso die Anzahl der eigenen Kanonen.
SPIDER MUTANTS ist ein – sagen wir einmal: – „ungewöhnliches" Spiel für den ZX-81. Es besticht durch seine Schnelligkeit und ist äußerst interessant. Die Redaktion meint, ein Spiel, das jeder ZX-81-Benutzer haben sollte! (M. K.)

Anleitung zum Abtippen:
Als erstes gibt man das kurze Programm zur Reservierung des Speicherplatzes für den Maschinencode ein. Dabei ist darauf zu achten, daß die REM-Zeile mindestens 25 Zeichen enthält. Hat man das Programm eingegeben, startet man es mit **„RUN"/„Newline"**. Nach kurzer Zeit erscheint auf dem Bildschirm **„1 REM"**. Nun gibt man **„GOTO 90"** und **„Newline"** ein. Wenn jetzt die Länge der REM-Zeile mit 4884 angegeben wird und das Programm mit der Fehlermeldung **„6/110"** abbricht, kann man die Zeilen 10 bis 110 löschen, damit nur noch eine REM-Zeile mit der Zeilennummer 0 übrig bleibt. Nun gibt man das Programm zur Eingabe des Maschinencodes ein, und startet es ebenfalls mit **„RUN"**. Jetzt gibt man zuerst die Startadresse des Maschinencodes ein und dann beliebig viele Bytes in hexadezimaler Schreibweise. Die Leerstelle zwischen den einzelnen Bytes kann auch weggelassen werden. Hat man die Bytes eingegeben, wartet das Programm auf einen Tastendruck, um dann wieder nach der Adresse zu fragen. Hat man so den ganzen Maschinencode eingegeben, kann das Eingabe-Programm gelöscht werden, so daß wieder nur die REM-Zeile mit der Zeilennummer 0 übrig bleibt, nur daß sie jetzt den Maschinencode enthält. Jetzt kann man das Basic-Listing zu „SPIDER MUTANTS" eingeben, und das Programm ist fertig. Vor dem ersten Programmstart sollte man sich eine Sicherheitskopie auf Cassette anfertigen, da das Programm abstürzen kann, wenn man den Maschinencode falsch eingegeben hat.



TEIL EINS

```
 10 REM ABCDEFGHIJKLMNOPQRSTUVWXYZ
 20 LET A$="0110132A1440361D2336EA2336762336762
    309CDB014C30C06"
 30 LET A=16514
 40 FOR N=1 TO LEN A$ STEP 2
 50 POKE A,16*(CODE A$(N)-28)+CODE A$(N+1)-28
 60 LET A=A+1
 70 NEXT N
 80 RAND USR 16514
 90 PRINT PEEK 16511+256*PEEK 16512
100 POKE 16510,0
110 PRINT 1/0
```

```
1150- 1160: 41366
1170- 1180: 69510
1190- 1200: 65728
1210- 1220: 25102
1230- 1240: 12768
1250- 1260: 18697
1270- 1280: 72211
1290- 1300: 25134
1310- 1320: 12768
1330- 1340: 43591
1350- 1360: 43203
1370- 1380: 72877
1390- 1400: 25166
1410- 1420: 12768
1430- 1440: 42858
1450- 1460: 70115
1470- 1480: 65220
1490- 1500: 25198
1510- 1520: 12768
1530- 1540: 41811
1550- 1560: 69340
1570- 1580: 66081
1590- 1600: 25230
1610- 1620: 12763
1630- 1640: 25813
1650- 1660: 79159
1670- 1680: 25262
1690- 1700: 12763
1710- 1720: 42303
1730- 1740: 74074
1750- 1760: 75061
1770- 1780: 25294
1790- 1800: 12763
1810- 1820: 41933
1830- 1840: 56491
1850- 1860: 89882
1870- 1880: 25223
1890- 1900: 12763
1910- 1920: 42226
1930- 1940: 65744
1950- 1960: 80179
1970- 1980: 23897
1990- 2000: 12768
2010- 2020: 25858
2030- 2040: 80218
2050- 2060: 23928
2070- 2080: 12768
2090- 2100: 42226
2110- 2120: 84050
2130- 2140: 78631
2150- 2160:  2764
2170- 2180: 37307
2190- 2200: 18970
2210- 2220:  1709
2230- 2240: 16895
2250- 2260: 16354
2270- 2280: 60637
2290- 2300: 91081
2310- 2320: 87953
2330- 2340: 81298
2350- 2360:  7642
2370- 2380: 40719
2390- 2400: 35739
2410- 2420: 20260
2430- 2440: 10024
2450- 2460: 50201
2470- 2480: 22351
2490- 2500:  4118
2510- 2520: 69804
2530- 2540: 88653
2550- 2560: 70449
2570- 2580: 71898
2590- 2600: 59459
2610- 2620: 26360
2630- 2640: 84726
2650- 2660:  8034
2670- 2680: 14665
2690- 2700: 22738
2710- 2720: 21100
2730- 2740: 50128
2750- 2760:  2964
2770- 2780: 73990
2790- 2800: 42896
2810- 2820: 35318
2830- 2840: 52822
2850- 2860: 56207
2870-     : 28368

GESAMTCHECKSUMME: 41005
```

Für die Freunde der Selbstverteidigung

„Thai-Boxing"

TI-99-Fans aufgepaßt! Dieses tolle Spiel stellt hohe Anforderungen an Körperbeherrschung, Konzentration und Reaktionsvermögen!

Nach dem Programmstart werdet ihr aufgefordert, einen Schwierigkeitsgrad in Form von drei wählbaren Gegnern sowie den eigenen Namen einzugeben. Danach findet ihr euch als weißen Thai-Boxer gemeinsam mit dem schwarzen Gegner in der Kampfarena wieder. Ihr müßt nun versuchen, dem Gegener möglichst viele Schläge und Tritte beizubringen. Doch der Kontrahent wehrt sich, denn er beherrscht alle Tritte und Schläge genauso perfekt wie ihr selbst.

Achtung: Bei Schwierigkeitsgrad 1 benötigt ihr oder der Gegener 5 Schläge oder Tritte, um einen von beiden auf die Matte zu legen. Bei den weiteren Schwierigkeitsgraden benötigt der Gegner nur 3 oder 4 Treffer, um euch zu bezwingen. Jedesmal, wenn ihr den Gegner auf den Boden befördert, wird ein Punkt gutgeschrieben. Sieger ist derjenige, der zuerst 10 Punkte erreicht.

Thai-Boxing wird über die Tastatur gesteuert. Die Tastenfunktionen können aus der Spielanleitung des Programmes entnommen werden. Zum Spiel muß die Alpha Lock-Taste gedrückt sein.

Und nun ran an die Tasten!

TEXAS INSTRUMENTS

```
10 CALL CLEAR :: CALL SPRITE(#1,84,16,30
,50,#2,72,2,30,80,#3,65,16,30,110,#4,73,
2,30,140)
20 CALL SPRITE(#5,66,16,50,50,#6,79,2,70
,70,#7,88,16,90,90,#8,73,2,110,110)
30 CALL SPRITE(#9,78,16,130,130,#10,71,2
,150,150)
40 DISPLAY AT(1,1):"INTERFACE-SOFTWARE P
RESENTS:" :: DISPLAY AT(24,3):"CO.1985 B
Y: DIRK BLUDAU"
50 DISPLAY AT(8,14)SIZE(15):"PRESS ANY K
EY !"
60 CALL MAGNIFY(1):: FOR I=1 TO 10 :: GO
SUB 70 :: NEXT I :: CALL MAGNIFY(2):: FO
R I=1 TO 10 :: GOSUB 70 :: NEXT I :: GOT
O 60
70 CALL KEY(0,K,S):: IF S<>0 THEN CALL C
LEAR :: CALL DELSPRITE(ALL):: GOTO 80 EL
SE RETURN
80 DISPLAY AT(3,3):"SPIELANLEITUNG ? <J>
<N>" :: FOR J=1 TO 100 :: NEXT J
90 CALL KEY(0,K,S):: IF S=0 THEN 90 ELSE
 IF K=74 THEN 1200 ELSE 100
100 Q,R=0 :: CALL CLEAR :: DISPLAY AT(3,
3):"SCHWIERIGKEITSGRAD ?" :: DISPLAY AT(
10,3):"1 = LEICHT (HO-CHI-MINH)"
```

Dieses Spiel ist lauffähig auf dem TI-99/4a Extended Basic

```
110 DISPLAY AT(12,3):"2 = NORMAL (CHANG-
PUANG)" :: DISPLAY AT(14,3):"3 = SCHWER
(CHAM-PI-IN)" :: ACCEPT AT(20,3)VALIDATE
("123")BEEP:C
120 IF C=1 THEN CW=5 ELSE IF C=2 THEN CW
=4 ELSE CW=3
130 CALL CLEAR :: CC=C*10 :: IF C=1 THEN
 CH=10 ELSE IF C=2 THEN CH=28 ELSE CH=46
140 DISPLAY AT(3,3):"GIB DEINEN NAMEN EI
N !" :: ACCEPT AT(10,3)VALIDATE(UALPHA)B
EEP SIZE(12):X$
150 IF C=1 THEN Y$="HO-CHI-MINH" ELSE IF
 C=2 THEN Y$="CHANG-PUANG" ELSE Y$="CHAM
-PI-IN"
160 CALL CLEAR :: DISPLAY AT(4,8)SIZE(12
):X$ :: DISPLAY AT(5,8)SIZE(12):Y$
170 CALL CHAR(33,"FFFFFFFFFFFFFFFF",34,"
00000000000000FF",35,"8080808080808080",
36,"FF000000000000000",37,"01010101010101
01")
180 CALL CHAR(40,"003C5A7E663C1818")
190 DATA 001C1A1C183C3F3C3F3C3C386CC6864
3000000000080000000000000000000000,001C1A
1C183C3F3C3F3C3C181818181C00000000008000
000000000000000000
200 DATA 000000000001000000000000000000
000385838183CFC3CFC3C3C1C366361C2,000000
000001000000000000000000000000385838183CFC
3CFC3C3C181818183B
210 DATA 000000000001030503010101030604 0
200E0D0E0C0E0FFE0E0E0E0C060303018,000101
010103070B0703030360C0C0600C0A0C080C8F0
C0C1FFFE0000000000
220 DATA 0101010103070B07037FFC800000000
0C0A0C080C0FCC0C1FF7E000000000000,00070B
070307FF070707070306 0C0C1800000000000 80C0
A0C0808080C0602040
230 DATA 000305030113 0F0383FF7F000000000
000808080C0E0D0E0E0E06030181830,030503
01033F0383FF7E000000000000808080C0E0D0
E0C0FE3F0100000000
240 DATA 000000000101010103070B07037FFE8
000000000C0A0C888C8F0C0C0F8FC0C06,000000
000000000000000000023577F7F00000000000000
0000000000C0E1FFFF
250 DATA 000000000305131113 0F03031F3F306
000000000808080C0E0D0E0C0FE7F01,000000
000000000000000000387FFFF00000000000000
0000000000C4EAFEFE
260 RESTORE 190 :: FOR I=88 TO 140 STEP
4 :: READ A$ :: CALL CHAR(I,A$):: NEXT I
 :: CALL COLOR(1,7,1,2,7,1):: RANDOMIZE
270 FOR I=9 TO 16 :: Q=Q+1 :: CALL HCHAR
(I,1,33,Q):: NEXT I
280 Q=33 :: FOR I=9 TO 16 :: Q=Q-1 :: R=
R+1 :: CALL HCHAR(I,Q,33,R):: NEXT I
290 CALL HCHAR(10,4,33,26):: Q=0 :: FOR
I=8 TO 15 :: Q=Q+1 :: CALL HCHAR(I,Q,40)
:: NEXT I :: Q=33 :: FOR I=8 TO 15 :: Q=
Q-1
300 CALL HCHAR(I,Q,40):: NEXT I :: CALL
HCHAR(9,4,40,26):: CALL HCHAR(8,5,40,24)
:: CALL HCHAR(7,6,40,22)
310 CALL VCHAR(1,8,37,6):: CALL VCHAR(1,
25,35,6):: CALL HCHAR(1,9,36,16):: CALL
HCHAR(6,9,34,16)
320 DISPLAY AT(2,9)SIZE(11):"PUNKTESTAND
" :: DISPLAY AT(4,20)SIZE(2):PU :: DISPL
AY AT(5,20)SIZE(2):GU
330 GOSUB 1300
340 CALL MAGNIFY(3):: CALL SPRITE(#1,88,
16,160,30,#2,96,2,160,226):: FOR I=1 TO
100 :: NEXT I :: RI=2 :: GI=2
350 CALL SOUND(200,-3,0):: CALL SOUND(20
0,-2,0)
360 GOSUB 690 :: CALL KEY(0,K,S):: IF S=
0 THEN 360
370 IF K=83 THEN X=-4 :: Y=0 :: RI=1 ::
GOTO 440
380 IF K=68 THEN X=4 :: Y=0 :: RI=2 :: G
OTO 450
390 IF K=44 THEN 460
400 IF K=46 THEN 510
410 IF K=76 THEN 560
420 IF K=32 THEN 1030
430 GOTO 360
440 CALL PATTERN(#1,100):: CALL MOTION(#
1,-Y*7,X*7):: CALL PATTERN(#1,96):: CALL
 MOTION(#1,0,0):: GOTO 360
450 CALL PATTERN(#1,92):: CALL MOTION(#1
,-Y*7,X*7):: CALL PATTERN(#1,88):: CALL
MOTION(#1,0,0):: GOTO 360
460 CALL SOUND(100,110,0,115,0,120,0)::
IF RI=1 THEN CALL PATTERN(#1,116):: GOTO
 470 ELSE CALL PATTERN(#1,104):: GOTO 49
0
470 FOR I=1 TO 50 :: NEXT I :: CALL COIN
C(ALL,CO):: IF CO=-1 THEN 480 ELSE CALL
PATTERN(#1,96):: GOTO 360
480 CALL PATTERN(#1,96):: IF GD>0 THEN G
OTO 360 ELSE GOTO 1060
490 FOR I=1 TO 50 :: NEXT I :: CALL COIN
C(ALL,CO):: IF CO=-1 THEN 500 ELSE CALL
PATTERN(#1,88):: GOTO 360
500 CALL PATTERN(#1,88):: IF GD>0 THEN G
OTO 360 ELSE GOTO 1060
510 CALL SOUND(100,110,0,115,0,120,0)::
IF RI=1 THEN CALL PATTERN(#1,120):: GOTO
 520 ELSE CALL PATTERN(#1,108):: GOTO 54
0
520 FOR I=1 TO 50 :: NEXT I :: CALL COIN
C(ALL,CO):: IF CO=-1 THEN 530 ELSE CALL
PATTERN(#1,96):: GOTO 360
530 CALL PATTERN(#1,96):: IF GD>0 THEN G
OTO 360 ELSE GOTO 1060
540 FOR I=1 TO 50 :: NEXT I :: CALL COIN
C(ALL,CO):: IF CO=-1 THEN 550 ELSE CALL
PATTERN(#1,88):: GOTO 360
550 CALL PATTERN(#1,88):: IF GD>0 THEN G
OTO 360 ELSE GOTO 1060
560 SP=1 :: IF RI=1 THEN 610 ELSE 570
570 CALL PATTERN(#1,112):: CALL MOTION(#
1,-20,20):: FOR I=1 TO 10 :: CALL COINC(
ALL,CO):: IF CO=-1 THEN GOTO 650 ELSE 58
0
580 NEXT I
590 CALL MOTION(#1,20,20):: FOR I=1 TO 1
0 :: CALL COINC(ALL,CO):: IF CO=-1 THEN
650 ELSE 600
600 NEXT I :: CALL PATTERN(#1,88):: CALL
```

```
 MOTION(#1,0,0):: CALL POSITION(#1,E,F):
: CALL LOCATE(#1,160,F):: GOTO 360
610 CALL PATTERN(#1,124):: CALL MOTION(#
1,-20,-20):: FOR I=1 TO 10 :: CALL COINC
(ALL,CO):: IF CO=-1 THEN GOTO 670 ELSE 6
20
620 NEXT I
630 CALL MOTION(#1,20,-20):: FOR I=1 TO
10 :: CALL COINC(ALL,CO):: IF CO=-1 THEN
 670 ELSE 640
640 NEXT I :: CALL PATTERN(#1,96):: CALL
 MOTION(#1,0,0):: CALL POSITION(#1,E,F):
: CALL LOCATE(#1,160,F):: GOTO 360
650 CALL SOUND(100,110,0,115,0,120,0):: 
CALL MOTION(#1,0,0):: CALL POSITION(#1,E
,F):: CALL LOCATE(#1,160,F):: CALL PATTE
RN(#1,88)
660 IF GD>0 THEN GOTO 360 ELSE 1060
670 CALL SOUND(100,110,0,115,0,120,0):: 
CALL MOTION(#1,0,0):: CALL POSITION(#1,E
,F):: CALL LOCATE(#1,160,F):: CALL PATTE
RN(#1,96)
680 IF GD>0 THEN GOTO 360 ELSE 1060
690 IF GD=1 THEN 1000 ELSE IF GD=2 THEN
1020 ELSE CALL POSITION(#1,E,F):: CALL P
OSITION(#2,EE,FF):: IF F<FF THEN 700 ELS
E 710
700 IF FF-F>CC THEN 730 ELSE IF SP=1 THE
N SP=0 :: CALL PATTERN(#2,88):: GI=1 ::
GOTO 720 ELSE GOTO 720
710 IF F-FF>CC THEN 740 ELSE IF SP=1 THE
N SP=0 :: CALL PATTERN(#2,96):: GI=2 ::
GOTO 720 ELSE GOTO 720
720 O=2+C :: ZU=INT(RND*O)+1 :: IF ZU=1
THEN 750 ELSE IF ZU=2 THEN 800 ELSE IF Z
U=3 THEN 980 ELSE 850
730 GI=1 :: CALL PATTERN(#2,100):: CALL
MOTION(#2,0,-CH):: CALL PATTERN(#2,96):: 
 CALL MOTION(#2,0,0):: RETURN
740 GI=2 :: CALL PATTERN(#2,92):: CALL M
OTION(#2,0,CH):: CALL PATTERN(#2,88):: C
ALL MOTION(#2,0,0):: RETURN
750 CALL SOUND(100,150,0,155,0,-2,0):: I
F GI=1 THEN CALL PATTERN(#2,116):: GOTO
760 ELSE CALL PATTERN(#2,104):: GOTO 780
760 FOR I=1 TO 50 :: NEXT I :: CALL COIN
C(ALL,CO):: IF CO=-1 THEN 770 ELSE CALL
PATTERN(#2,96):: RETURN
770 CALL PATTERN(#2,96):: IF GE=1 THEN R
ETURN ELSE GOTO 1090
780 FOR I=1 TO 50 :: NEXT I :: CALL COIN
C(ALL,CO):: IF CO=-1 THEN 790 ELSE CALL
PATTERN(#2,88):: RETURN
790 CALL PATTERN(#2,88):: IF GE=1 THEN R
ETURN ELSE GOTO 1090
800 CALL SOUND(100,150,0,155,0,-2,0):: I
F GI=1 THEN CALL PATTERN(#2,120):: GOTO
810 ELSE CALL PATTERN(#2,108):: GOTO 830
810 FOR I=1 TO 50 :: NEXT I :: CALL COIN
C(ALL,CO):: IF CO=-1 THEN 820 ELSE CALL
PATTERN(#2,96):: RETURN
820 CALL PATTERN(#2,96):: IF GE=1 THEN R
ETURN ELSE GOTO 1090
830 FOR I=1 TO 50 :: NEXT I :: CALL COIN
C(ALL,CO):: IF CO=-1 THEN 840 ELSE CALL
PATTERN(#2,88):: RETURN
840 CALL PATTERN(#2,88):: IF GE=1 THEN R
ETURN ELSE GOTO 1090
850 IF GI=1 THEN 900 ELSE 860
860 CALL PATTERN(#2,112):: CALL MOTION(#
2,-20,20):: FOR I=1 TO 10 :: CALL COINC(
ALL,CO):: IF CO=-1 THEN GOTO 940 ELSE 87
0
870 NEXT I
880 CALL MOTION(#2,20,20):: FOR I=1 TO 1
0 :: CALL COINC(ALL,CO):: IF CO=-1 THEN
940 ELSE 890
890 NEXT I :: CALL PATTERN(#2,88):: CALL
 MOTION(#2,0,0):: CALL POSITION(#2,E,F):
: CALL LOCATE(#2,160,F):: RETURN
900 CALL PATTERN(#2,124):: CALL MOTION(#
2,-20,-20):: FOR I=1 TO 10 :: CALL COINC
(ALL,CO):: IF CO=-1 THEN GOTO 960 ELSE 9
10
910 NEXT I
920 CALL MOTION(#2,20,-20):: FOR I=1 TO
10 :: CALL COINC(ALL,CO):: IF CO=-1 THEN
```

„Cheng-Wang, ich freue mich ja wirklich, wenn Deine Brüder uns besuchen. Aber können sie nicht wie jeder andere an die Tür klopfen?"

```
 960 ELSE 930
930 NEXT I :: CALL PATTERN(#2,96):: CALL
 MOTION(#2,0,0):: CALL POSITION(#2,E,F):
: CALL LOCATE(#2,160,F):: RETURN
940 CALL SOUND(100,150,0,155,0,-2,0):: C
ALL MOTION(#2,0,0):: CALL POSITION(#2,E,
F):: CALL LOCATE(#2,160,F):: CALL PATTER
N(#2,88)
950 IF GE=1 THEN RETURN ELSE 1090
960 CALL SOUND(100,150,0,155,0,-2,0):: C
ALL MOTION(#2,0,0):: CALL POSITION(#2,E,
F):: CALL LOCATE(#2,160,F):: CALL PATTER
N(#2,96)
970 IF GE=1 THEN RETURN ELSE 1090
980 IF GI=2 THEN 990 ELSE 1010
990 GD=1 :: CALL PATTERN(#2,128):: CALL
KEY(0,K,S):: GOTO 370:
1000 GD=0 :: CALL PATTERN(#2,88):: CALL
KEY(0,K,S):: GOTO 370
1010 GD=2 :: CALL PATTERN(#2,136):: CALL
 KEY(0,K,S):: GOTO 370
1020 GD=0 :: CALL PATTERN(#2,96):: CALL
KEY(0,K,S):: GOTO 370
```

```
1030 GE=1 :: IF RI=2 THEN 1040 ELSE 1050
1040 CALL PATTERN(#1,128):: GOSUB 690 ::
 GE=0 :: CALL PATTERN(#1,88):: CALL KEY(
0,K,S):: GOTO 370
1050 CALL PATTERN(#1,136):: GOSUB 690 ::
 GE=0 :: CALL PATTERN(#1,96):: CALL KEY(
0,K,S):: GOTO 370
1060 P=P+1 :: IF P=5 THEN 1070 ELSE GOTO
 360
1070 P,W=0 :: IF GI=1 THEN CALL PATTERN(
#2,132)ELSE CALL PATTERN(#2,140)
1080 CALL SOUND(1300,-5,8):: FOR I=1 TO
100 :: NEXT I :: PU=PU+1 :: DISPLAY AT(4
,20)SIZE(3):PU :: IF PU=10 THEN GOTO 112
0 ELSE GOTO 340
1090 W=W+1 :: IF W=CW THEN 1100 ELSE CAL
L KEY(0,K,S):: GOTO 370
1100 W,P=0 :: IF RI=2 THEN CALL PATTERN(
#1,132)ELSE CALL PATTERN(#1,140)
1110 CALL SOUND(1300,-5,8):: FOR I=1 TO
100 :: NEXT I :: GU=GU+1 :: DISPLAY AT(5
,20)SIZE(3):GU :: IF GU=10 THEN GOTO 114
0 ELSE 340
1120 PU,GU=0 :: FOR I=1 TO 20 :: DISPLAY
 AT(4,8)SIZE(12):"" :: FOR VER=1 TO 20 :
: NEXT VER :: DISPLAY AT(4,8)SIZE(12):X$
 :: NEXT I
1130 GOSUB 1330 :: GOTO 1160
1140 GU,PU=0 :: FOR I=1 TO 20 :: DISPLAY
 AT(5,8)SIZE(12):"" :: FOR VER=1 TO 20 :
: NEXT VER :: DISPLAY AT(5,8)SIZE(12):Y$
 :: NEXT I
1150 FOR I=600 TO 300 STEP -30 :: CALL S
OUND(550,I,2,I-20,6,I+10,5):: NEXT I
1160 CALL DELSPRITE(ALL)
1170 DISPLAY AT(12,3)SIZE(20):"NEUES SPI
EL ? <J><N>" :: CALL KEY(0,K,S):: IF S=0
 THEN 1170 ELSE IF K=74 THEN 100 ELSE IF
 K=78 THEN 1190
1180 GOTO 1170
1190 CALL CLEAR :: END
1200 CALL CLEAR :: PRINT "THAI - BOXING"
1210 PRINT "ALS THAI-BOXER TRETEN SIE
GEGEN EINEN VON DREI WAEHL- BAREN GEGNER
N AN.WENN SIE"
1220 PRINT "IHREM GEGNER MEHRERE SCHLAE-
GE ODER TRITTE VERABREICHT  HABEN,DANN H
ABEN SIE IHN AUF"
1230 PRINT "DIE MATTE GELEGT UND BEKOMM-
EN  EINEN PUNKT.DAS GILT NA-TUERLICH AUC
H UMGEKEHRT.WER"
1240 PRINT "ZUERST 10 PUNKTE HAT,IST DER
SIEGER DES KAMPFES.          STEUERUNG DE
R SPIELFIGUR:"
1250 PRINT "S - NACH LINKS GEHEN
D - NACH RECHTS GEHEN        , - BOXEN"
1260 PRINT ". - TRETEN
L - SPRINGEN                 LEERTASTE -
ABWEHRSTELLUNG"
1270 PRINT "DRUECKEN SIE EINE TASTE !!!"
1280 CALL KEY(0,K,S):: IF S=0 THEN 1280
ELSE 100
1290 DATA 165,180,232,232,287,346,408,40
8,474,287,232,287,346,544,474
1300 RESTORE 1290 :: FOR I=1 TO 15 :: RE
AD Z
1310 CALL SOUND(+200,Z,0):: NEXT I :: RE
TURN
1320 DATA 233,246,261,246,277,261,277,29
3,311,293,277,293,311,261,246,277,261,29
3,311,329,349
1330 RESTORE 1320 :: FOR I=1 TO 21 :: RE
AD Z :: CALL SOUND(100,Z,0,349,5):: NEXT
 I :: RETURN
```

THAI BOXING — THE END

Letzte Meldung! Letzte Meldung!

ENTERPRISE hat die Preise für seine Super-Rechner gesenkt!

Der 64K-Computer kostet jetzt ganze 498 Mark, während der 128K-Enterprise schon für 798 Mark zu haben ist. Die Peripherie (ebenfalls im Preis drastisch gesenkt: Monitor, grün – 298,– / Monitor, farbig (Hi-Res) – 798,– / Drucker – 898,– / 3,5 Zoll Laufwerk mit Controller – 998,– / zweites Laufwerk ohne Controller – 698,– (720 KB-formatiert). Klasse Preise für einen Riesen-Computer!

Korrektur zum Programm „Brückenbau" für den VC-20 COMPUTRONIC 1/2/86

```
8 C$(1)="%%%%%%%%%%%%%%%":C$(2)="%%%%%%%%%%%%%%%":G$="[illegible]"

49 IFPEEK(Z+26)=26ORPEEK(Z+26)=27ORPEEK(Z+27)=26THENPOKEZ,32:POKEZ+1,32:W=0:RETU
RN
```

Tips & Tricks für den TI-99/4A

Trotz der Produktionseinstellung des TI-99 erfreut sich der Computer immer noch einer großen Beliebtheit. Vor allem Einsteiger legen sich oft einen preiswerten, meist gebrauchten, TI-99/4A zu. Deshalb werden folgende Tips & Tricks bei unseren Lesern sicherlich reges Interesse finden:

1. Im Command-Modus, also bei der Eingabe von Programmen, kann man die Bildschirmfarbe normalerweise nicht ändern, weil ein „CALL SCREEN" nur eine Farbänderung für wenige Sekunden erzeugt, danach ist der Bildschirm wieder hellblau. Folgender Trick hingegen bewirkt eine dauerhafte Änderung:
   CALL SCREEN(2):: CALL COLOR(0,8,2):: FOR Z=2 TO 8:: CALL COLOR(Z,8,2):: NEXT Z:: ACCEPT A$
   Wenn diese Befehlsfolge im Commandmodus (d. h. ohne Zeilennummer) eingegeben wird, färbt sich der Bildschirm schwarz und die Schrift hellblau. Nun kann ein Programm eingegeben werden, allerdings muß die ENTER-Taste zunächst noch einmal gedrückt werden.
   Bei der Eingabe eines CALL-Kommandos oder bei einer falschen Eingabe nimmt der Bildschirm wieder seine ursprüngliche Farbe an.

2. Der Extended Basic Befehl SIZE kann in TI Basic folgendermaßen simuliert werden:
   30000 Z=Z+8
   30010 GOSUB 30000
   Nachdem RUN eingegeben wurde, ruft sich das Unterprogramm immer wieder selbst auf, bis schließlich eine MEMORY FULL-Meldung ausgegeben wird. Nun gegen Sie ein : PRINT Z;"Bytes free"
   Der noch freie Speicherplatz wird ausgegeben.

3. Das Extended Basic gestattet den Befehl ON BREAK NEXT. Dazu zwei wichtige Tips: Wenn Sie ein Programm entwikkeln, vernachlässigen Sie diesen Befehl zunächst, denn es könnte je noch ein Fehler im Programm enthalten sein. Falls sich der Computer nun aber in einer Endlosschleife aufhängen sollte, so kann das Programm nicht mehr mit FCTN 4 abgebrochen werden. Es muß mit QUIT gelöscht werden! Daher sollte man den ON BREAK NEXT-Befehl erst in das Programm einfügen, wenn es wirklich fehlerfrei läuft. Zusätzlich sollten Sie beachten, daß der Befehl wirkungslos wird, falls das Programm mit „RUN X" ab der Zeile X neu gestartet wird (z. B. bei Spielen) und falls der Befehl eine Zeilennummer kleiner X hat. In diesem Fall sollte man den ON BREAK NEXT Befehl in der Zeile X noch einmal eingeben.

4. Einige TI-Freunde kennen vielleicht das Texas Instruments Lernspiel „Speak & Spell". Die Regeln sind schnell erklärt: Der Computer spricht 10 englische Worte aus und der Spieler muß die Worte richtig eingeben. Es soll Englisch-Lernenden beim Buchstabieren der Wörter helfen. Dieses Spiel läßt sich auf dem TI-99 hervorragend simulieren, falls der Sprachsynthesizer vorhanden ist:
   100 - 200 DATAs : Hier geben Sie die Worte ein, die der Sprachsynthesizer aussprechen kann (Liste liegt der Anleitung bei!)
   210 FOR Z=1 TO ANZAHL (Anstelle von ANZAHL geben Sie die Zahl der Wörter ein, die Sie in die DATAs übernommen haben.)
   220 READ A$(Z):: NEXT Z
   230 FOR Z=1 TO 10:: B=INT(RND*Anzahl)+1:: CALL SAY(A$(B)):: INPUT B$
   240 IF B$=A$(B) THEN CALL SAY("-RIGHT") ELSE F=F+1:: PRINT A$(B):: CALL SOUND(100,110,0)
   250 NEXT Z:: PRINT F;" FEHLER"
   Sollten Sie mehr als 10 Worte in den DATAs eingegeben haben (was ja auch nur sinnvoll ist), dann fügen Sie noch den Befehl 10 DIM A$(Anzahl) ein, wobei ANZAHL wieder die Zahl der eingegebenen Worte ist. Falls ein falsches Wort eingegeben wurde, zeigt der Computer die richtige Schreibweise an.

5. Folgender Befehl läßt den TI-99 abstürzen: CALL PEEK(-26000,A)
   Verwendung z. B. bei einem Listschutz.

6. Folgende Befehlsfolge wird oft gebraucht, um ein Programm erst beim Druck der ENTER Taste weiterlaufen zu lassen :
   100 CALL KEY(O,K,S):: IF S=O THEN 100
   110 IF K=13 then 120 else 100
   120 (hier geht das Programm weiter)
   Der folgende Trick erreicht das gleiche viel einfacher:
   100 ACCEPT AT(24,1)VALIDATE("")
   SIZE(O):A$

7. In Extended Basic kann man sich bei einigen Befehlen das "::" !" (und damit Speicherplatz) sparen; z. B. bei STOP, END, GOTO X, GOSUB X, etc. So können Bemerkungen einfach an den Befehl angefügt werden :
   100 GOTO 200 VERZWEIGUNG
   200 END PROGRAMMENDE usw.

IMPRESSUM

Computronic
erscheint alle 2 Monate im Tronic-Verlag, 3444 Wehretal 1
Tel. 0 56 51 / 4 06 93 oder 4 06 43
3. Jahrgang

Herausgeber: Axel Credé
Chefredakteur: Siegfried Görk
Redaktion: Manfred Kleimann, Hartmut Wendt, Frank Brall, Ottfried Schmidt, Bernd Zimmermann

Programmautoren: R. Krotz, J. Rindsfüßer, J. Anton, G. Tentler, O. Schmidt, M. Droste, H. Spierling, M. Eckhardt, T. Goesmann, J. Tuttas, U. Weier, T. Henricksmann, J. Hempeler, K. Kiewert, M. Potsch, O. Cyranka, O. Steinmeier, K. Ezcan u. einige andere.

Gesamtherstellung:
Druckhaus Dierichs Kassel, Frankfurter Str. 168, 3500 Kassel

Vertrieb:
Inland (Groß-, Einzel- und Bahnhofsbuchhandel) sowie Österreich und Schweiz:
Verlagsunion
Friedrich-Bergius-Straße 20
6200 Wiesbaden
Telefon 0 61 21 / 26 60

Anzeigenverwaltung (Inland):
Hartmut Wendt, Tronic-Verlag, Postfach 41, 3444 Wehretal 1, Tel. 0 56 51 / 4 06 93

Anzeigenleitung (Ausland):
Public Relation (Foreign Countries):
Manfred Kleimann, c/o Tronic-Verlag, Postfach 41 (Post Box 41)
D-3444 Wehretal 1, Tel. 0 56 51 / 4 06 43

Erscheinungsweise: Erstverkaufstag von COMPUTRONIC ist Anfang des Monats

Anzeigenpreisliste:
Bitte Mediaunterlagen anfordern.

Software-Service und Vertrieb: Annelie Kratzenberg, Heike Rabe

Urheberrecht:
Alle in COMPUTRONIC veröffentlichten Beiträge sind urheberrechtlich geschützt. Alle Rechte, auch Übersetzungen, vorbehalten.
Reproduktionen jeder Art (Fotokopien, Microfilm, Erfassung in Datenverarbeitungsanlagen usw.) bedürfen der schriftlichen Genehmigung des Verlags. Alle veröffentlichte Software wurde von Mitarbeitern des Verlages oder von freien Mitarbeitern erstellt.
Aus Ihrer Veröffentlichung kann nicht geschlossen werden, daß die beschriebenen Lösungen oder Bezeichnungen frei von Schutzrechten sind.

Bezugspreise:
Einzelheft 6,50 DM

Autoren, Manuskripte:
Der Verlag nimmt Manuskripte und Software zur Veröffentlichung gerne entgegen. Sollte keine andere Vereinbarung getroffen sein, so gehen wir davon aus, daß Sie mit einem Honorar von 120,– DM pro abgedruckter Seite im Heft einverstanden sind.

Titelblatt: Werbeatelier H. Kästle, Eschwege

Mit den Schlümpfen auf einer interessanten aber auch gefährlichen Wanderung

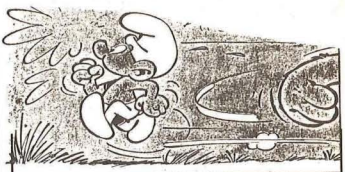
Ein Geschicklichkeitsspiel für den TI-99

Wanderer

Steuern Sie Ihre Schlümpfe sicher und erfolgreich über die vorgegebene Wanderstrecke und überwinden Sie die zahlreichen Hindernisse. Ein TI-Programmbeitrag der wiederum Action, schnelles Reagieren, Anpassung an die Art der Steuerung und ein optimales Timing erfordert.

Nach dem erfolgreichen Eintippen starten Sie das Programm mit „RUN",e und das Titelbild erscheint. Wenn Sie anschließend den Feuerknopf drücken, erreichen Sie das 1. Bild des Spiels.
Es ist nun Ihre Aufgabe, die dort befindlichen Gegenstände mit dem Schlumpf zu überspringen. Die Steuerung ist etwas unüblich und bedarf daher einiger Übung!
Sie besitzen am Anfang 2 000 Punkte. Jeder Zusammenstoß mit einem der Hindernisse (Baumstumpf, Pfütze, Grube) kostet Sie 20 Punkte. Kommt es zu einer Kollision mit einem ganz speziellen Kriechtier, verlieren Sie eines Ihrer fünf Leben. Sollten Sie nach dem 6. Bild noch mehr als 1 800 Punkte aufweisen, bekommen Sie einen Bonus von 2 000 Punkten. Nach jeweils 9 weiteren Bildern kann, falls nicht zu viele Spielfehler begangen wurden, ein weiterer Bonus erreicht werden. Nach dem erfolgreichen Absolvieren von 9 Bildern wiederholen sich die vorhergehenden in zufälliger Reihenfolge.

STEUERUNG



„Wanderer" präsentiert Ihnen Schlümpfe in Action!

Die Steuerung erfolgt per Joystick: Joystick nach links oder rechts bewegt den Schlumpf nach vorn (rückwärts nicht möglich), Druck auf den Feuerknopf bewirkt einen Standsprung, Druck des Feuerknopfes und Joystick nach rechts oder links bewirkt einen Sprung nach vorn.

WANDERER — Das Listing — WANDERER — Das Listing

```
130 RANDOMIZE
140 JO=2
150 PKT=2000 :: LIVE=5 :: BONI=1800
160 CP=-8
170 DIM BILD(9)
180 FOR I=1 TO 9 :: BILD(I)=I :: NEXT I
190 CALL CLEAR :: CALL MAGNIFY(3):: CALL
 SCREEN(1):: CALL COLOR(1,4,4):: CALL CH
AR(33,"FFFFFFFFFFFFFFFF"):: CALL CHAR(32
,"0")
200 FOR I=1 TO 14 :: CALL COLOR(I,4,4)::
 NEXT I
210 ! HAUS
220 DATA 0101010303070F1F,808080C0C0E0F0
F8,3F0F0F1C1C1F1FDF,FCF0F03838F8F8FB,010
1010303030303
230 DATA DFDFFFFFF9FFF9FF,FBFBFFFF3FFF3F
FF,80808080C0C0C0C0,0707070F0F0F0F0F
240 DATA FFFFD7D7FFFFFFFF,E0E0E0F0F0F0F0
F0
250 DATA FFFFE3C1C5C1C1FF
260 ! SCHLUMPF (STEHT)
270 DATA 00000000030F070707070F0F0000000
000000000C0E0E0800080C0E000000000
280 DATA 03070F0F0C000000000000000F07030
7E0C080C000800000000000000C08000C0
290 ! SCHLUMPF (GEHT RECHTS)
300 DATA 00070F0F0C000000000000000F03070
400E080C000800000000000000C0804080
310 ! HINDERNISSE
320 DATA 000003070F17171B1C1F1F3F3F7F7FF
F0000E0F0F8FCF4EC1CFCFCFCFEFEFEFF
330 DATA 00000000001F7FFFFFFFFF7F3F1F070
000000000003CFEFFFFFFFFFEFCE0C000
340 DATA 0000000000008060983F3F3F9A65820
100000602121F030303FBFFFCA040A050
350 DATA 0000000E1F312003060D0B0B0B05030
000464E9CB8FEC0C0E0F0F0F0E0C08000
360 ! WIESE BAUM
370 DATA 0000000000AAAAFF,FFFFFFFFFFFFFF
FF,183C7E7E7E7E7E3C
380 ! ZAUN STAMM
390 DATA 80FF808080FF8080,1818183C000000
00
400 ! GEBIRGE
410 DATA FFFFFFFFFFFFFFFFFF,0103070F1F3F7
FFF,80C0E0F0F8FCFEFF
420 ! GEBIRGSPOSITIONEN
430 DATA 8,1,7,2,8,5,8,7,7,8,7,10,6,11,5
,12,7,16,8,19,7,20,8,23,7,24,6,25,5,26,6
,29
440 DATA 7,3,8,4,8,6,7,9,5,13,6,14,7,15,
7,17,8,18,7,21,8,22,5,27,6,28,6,30,7,31,
8,32
450 ! TITELSCHRIFT SMURF
460 DATA 00E262723139191D0C0E07000000000
000232327464E4C5C98B1E30306060C0C
470 DATA 00800006050666F3F39B0D9D6606060
60003010509884850303 0B091D1D1D1D3
480 DATA 008F66339391898 98B8B9396362C6C9
F70F0000C07A6E3E323030706060C2CEC
490 DATA 0000301F985C2626C60303834626462
F0000408000100E0949CDCE4C0E0959BA
500 CALL CHARPAT(73,A$):: IF A$="0103070
F1F3F7FFF" THEN 520 ELSE 510
510 CALL TITEL
520 RESTORE 270 :: FOR I=132 TO 140 STEP
 4 :: READ A$ :: CALL CHAR(I,A$):: NEXT
I
530 RESTORE 320 :: READ A$ :: CALL CHAR(
128,A$)
540 CALL NEUBILD(BILD(),PKT,BONI)
550 CALL COLOR(14,5,13,13,5,13)
560 FOR I=6 TO 9 :: CP=CP+24 :: CALL SPR
```

```
ITE(#I,136,16,170,CP):: NEXT I
570 FOR I=3 TO 14 STEP 3
580 CALL HCHAR(22,I,132):: CALL HCHAR(22
,I+1,134):: CALL HCHAR(23,I,133):: CALL
HCHAR(23,I+1,135):: NEXT I
590 CALL SPRITE(#1,136,16,144,5,#2,132,5
,144,5)
600 DISPLAY AT(1,10)SIZE(6):PKT
610 CALL KEY(JO,K,S):: IF K=18 THEN CALL
 SPRUNG(X,PKT,LIVE)
620 CALL JOYST(JO,X,Y)
630 CALL COINCI(PKT,LIVE)
640 CALL PATTERN(#1,136)
650 CALL POSITION(#1,P1,P2):: IF P2>230
THEN CALL NEUBILD(BILD(),PKT,BONI)
660 CALL MOTION(#2,0,((X*X)/4)*3,#1,0,((
X*X)/4)*3)
670 IF X<>0 THEN CALL PATTERN(#1,140)::
GOTO 610
680 GOTO 610
690 SUB BONUS(PKT)
700 CALL DELSPRITE(#1,#2,#3,#4,#5)
710 RESTORE 350 :: READ A$ :: CALL CHAR(
88,A$)
720 CALL SPRITE(#15,88,9,144,100)
740 CALL SPRITE(#1,136,16,144,5,#2,132,5
,144,5)
750 CALL JOYST(2,X,Y)
760 CALL COINC(#1,#15,16,ZE):: IF ZE<>0
THEN 770 ELSE 800
770 CALL MOTION(#1,0,0,#2,0,0):: FOR I=1
 TO 8 :: CALL SOUND(10,-7,0):: FOR Y=1 T
O 10 :: NEXT Y :: NEXT I :: CALL SOUND(-
45,-6,0):: PKT=PKT+2000
780 DISPLAY AT(1,10)BEEP SIZE(6):PKT
790 CALL DELSPRITE(#15)
800 CALL PATTERN(#1,136)
810 CALL POSITION(#1,A1,B2):: IF B2>230
THEN CALL DELSPRITE(#15):: SUBEXIT
820 CALL MOTION(#2,0,((X*X)/4)*3,#1,0,((
X*X)/4)*3)
830 IF X<>0 THEN CALL PATTERN(#1,140)::
GOTO 750
840 GOTO 750
850 SUBEND
860 SUB TITEL
870 RESTORE 370
880 FOR I=104 TO 106 :: READ ZEI$ :: CAL
L CHAR(I,ZEI$):: NEXT I
890 RESTORE 410
900 FOR I=112 TO 114 :: READ ZEI$ :: CAL
L CHAR(I,ZEI$):: NEXT I
910 CALL COLOR(10,13,4,11,3,8,1,4,8,3,2,
13,4,2,13,5,2,13,6,2,13,7,2,13,8,2,13)
920 CALL HCHAR(9,1,33,352):: CALL HCHAR(
20,1,105,160):: CALL HCHAR(19,1,104,32)
930 CALL HCHAR(8,1,112,32):: CALL HCHAR(
7,11,112,4):: CALL HCHAR(6,12,112,2):: C
ALL HCHAR(7,25,112,6):: CALL HCHAR(6,26,
112,2)
940 RESTORE 430
950 FOR I=1 TO 16 :: READ A,B :: CALL HC
HAR(A,B,113):: NEXT I
960 FOR I=1 TO 16 :: READ A,B :: CALL HC
HAR(A,B,114):: NEXT I
970 RESTORE 460
980 FOR I=116 TO 128 STEP 4 :: READ A$ :
: CALL CHAR(I,A$):: NEXT I
990 DISPLAY AT(22,1)SIZE(31):"iiiCOPYRIG
HTi1985iBYiGURKEii"
1000 CALL SPRITE(#20,116,2,100,96,#21,12
0,2,100,112,#22,124,2,100,128,#23,128,2,
100,144)
1010 FOR I=1 TO 20 :: CALL COLOR(#20,16,
#21,16,#22,16,#23,16)
1020 FOR TY=1 TO 50 :: CALL KEY(2,K,S)::
 IF K=18 THEN 1070
1030 NEXT TY
1040 CALL COLOR(#20,2,#21,2,#22,2,#23,2)
1050 FOR TY=1 TO 50 :: CALL KEY(2,K,S)::
 IF K=18 THEN 1070
1060 NEXT TY :: NEXT I :: GOTO 1010
1070 CALL SOUND(50,-1,0)
1080 CALL CLEAR :: FOR I=1 TO 14 :: CALL
 COLOR(I,4,4):: NEXT I :: CALL DELSPRITE
(ALL)
1090 SUBEND
1100 SUB ONE
1110 RESTORE 370
1120 FOR I=40 TO 42 :: READ A$ :: CALL C
HAR(I,A$):: NEXT I
1130 RESTORE 390 :: FOR I=64 TO 65 :: RE
AD A$ :: CALL CHAR(I,A$):: NEXT I
1140 RESTORE 410 :: FOR I=72 TO 74 :: RE
AD A$ :: CALL CHAR(I,A$):: NEXT I
1150 CALL HCHAR(8,1,72,32):: CALL HCHAR(
7,11,72,4):: CALL HCHAR(6,12,72,2):: CAL
L HCHAR(7,25,72,6):: CALL HCHAR(6,26,72,
2)
1160 FOR I=1 TO 16 :: READ A,B :: CALL H
CHAR(A,B,73):: NEXT I :: FOR I=1 TO 16 :
: READ A,B :: CALL HCHAR(A,B,74):: NEXT
I
1170 RESTORE 330 :: READ A$ :: CALL CHAR
(124,A$)
1180 CALL COLOR(2,13,4,1,4,8,5,7,4,3,16,
8,4,16,8,6,13,8)
1190 CALL HCHAR(9,1,33,352):: CALL HCHAR
(20,1,41,160):: CALL HCHAR(20,1,40,32)::
 CALL HCHAR(17,20,64,13)
1200 CALL HCHAR(15,5,42,9):: CALL HCHAR(
16,5,65,9):: CALL HCHAR(13,2,42,9):: CAL
L HCHAR(14,2,65,9)
1210 CALL HCHAR(11,1,42,5):: CALL HCHAR(
12,1,65,5)
1220 CALL SPRITE(#3,128,2,144,190)
1230 CALL SPRITE(#5,124,8,150,90)
1240 SUBEND
1250 SUB TWO
1260 CALL DELSPRITE(#1,#2,#3,#4,#5)
1270 CALL HCHAR(9,1,33,352)
1280 CALL HCHAR(13,23,42,9):: CALL HCHAR
(14,23,65,9):: CALL HCHAR(15,18,42,9)::
CALL HCHAR(16,18,65,9)
1290 CALL HCHAR(16,4,42,5):: CALL HCHAR(
17,4,65,5)
1300 CALL SPRITE(#3,128,2,144,90,#5,128,
2,144,150)
1310 CALL SPRITE(#1,136,16,144,5,#2,132,
5,144,5)
1320 FOR I=1 TO 300 :: NEXT I
1330 SUBEND
1340 SUB THREE
```

```
1350 CALL DELSPRITE(#1,#2,#3,#4,#5):: CA
LL COLOR(8,7,4,9,2,4,10,2,4,12,2,11)
1360 CALL HCHAR(9,1,33,352)
1370 RESTORE 220 :: FOR I=94 TO 104 :: R
EAD A$ :: CALL CHAR(I,A$):: NEXT I :: RE
AD A$ :: CALL CHAR(122,A$):: RESTORE 340
 :: READ A$ :: CALL CHAR(108,A$)
1380 DISPLAY AT(9,3):"!!!!!!!!!!!!!!!!!
!!^_!!!!!!!!!!!!!!!!!!!!!!!!!'a!!!!!!!!
!!!!!!!!!!!!!!!!bcde!!!!!!!!!!!!!!!!!!!
!!!!!fzgh!";
1390 CALL HCHAR(9,1,33,32)
1400 CALL HCHAR(15,3,42,9):: CALL HCHAR(
16,3,65,9):: CALL HCHAR(13,10,42,9):: CA
LL HCHAR(14,10,65,9):: CALL HCHAR(16,23,
64,10)
1410 CALL HCHAR(12,29,42,4):: CALL HCHAR
(13,29,65,4):: CALL HCHAR(14,28,42,4)::
CALL HCHAR(15,28,65,4)
1420 CALL SPRITE(#4,108,16,142,190,#5,12
4,4,150,125,#3,128,2,144,75)
1430 CALL SPRITE(#1,136,16,144,5,#2,132,
5,144,5)
1440 FOR I=1 TO 300 :: NEXT I
1450 SUBEND
1460 SUB FOUR
1470 CALL DELSPRITE(#1,#2,#3,#4,#5)
1480 CALL HCHAR(9,1,33,352)
1490 CALL HCHAR(18,1,64,32)
1500 CALL HCHAR(12,4,42,4):: CALL HCHAR(
13,4,65,4)
1510 CALL HCHAR(13,25,42,3):: CALL HCHAR
(14,25,65,3)
1520 CALL HCHAR(14,12,42,2):: CALL HCHAR
(15,12,65,2)
1530 CALL HCHAR(12,17,42):: CALL HCHAR(1
3,17,65)
1540 CALL SPRITE(#3,128,2,144,88,#4,108,
16,144,240,0,-6)
1550 CALL SPRITE(#1,136,16,144,5,#2,132,
5,144,5)
1560 SUBEND
1570 SUB FIVE
1580 CALL DELSPRITE(#1,#2,#3,#4,#5)
1590 CALL HCHAR(9,1,33,352)
1600 CALL HCHAR(16,1,64,15)
1610 CALL HCHAR(11,29,42,3):: CALL HCHAR
(12,29,65,3)
1620 CALL HCHAR(13,27,42,6):: CALL HCHAR
(14,27,65,6)
1630 CALL HCHAR(15,23,42,10):: CALL HCHA
R(16,23,65,10)
1640 CALL HCHAR(12,22,42):: CALL HCHAR(1
3,22,65)
1650 CALL HCHAR(12,8,42,2):: CALL HCHAR(
13,8,65,2)
1660 CALL SPRITE(#3,128,2,144,104,#5,124
,4,150,180,#4,108,16,142,240,0,-6)
1670 CALL SPRITE(#1,136,16,144,5,#2,132,
5,144,5)
1680 SUBEND
1690 SUB SIX
1700 CALL DELSPRITE(#1,#2,#3,#4,#5)
1710 CALL HCHAR(9,1,33,352)
1720 DISPLAY AT(9,3):"!!!!!!!!!!!!!!!!!!
!!^_!!!!!!!!!!!!!!!!^_!!!!!!!!!'a!!!!!!!!
!!!!!!!'a!!!!***!bcde!!*****!!!**!bcde!!
!AAA!fzgh!!AAAAA!!!AA!fzgh!!!!!!!!!!!!!
!!";
1730 CALL HCHAR(9,1,33,32):: CALL HCHAR(
18,24,64,9):: CALL HCHAR(17,26,65,6)
1740 CALL HCHAR(15,2,42,5):: CALL HCHAR(
16,2,65,5):: CALL HCHAR(17,16,42):: CALL
 HCHAR(18,16,65):: CALL HCHAR(16,26,42,6
)
1750 CALL SPRITE(#1,136,16,144,5,#2,132,
5,144,5)
1760 CALL SPRITE(#3,128,2,144,90,#5,124,
8,150,140)
1770 SUBEND
1780 SUB SEVEN
1790 CALL DELSPRITE(#1,#2,#3,#4,#5)
1800 CALL HCHAR(9,1,33,352)
1810 DISPLAY AT(9,3):"!!!!!!!*!!*!!!!***
**^_***!!****!!!!A!!A!!!AAAAA'aAAA!!AAA
A!!****!!!*****!!bcde*****!!!!AAAA!!!AA
AAA!!fzghAAAAA";
1820 CALL HCHAR(9,1,33,32)
1830 CALL HCHAR(14,3,42,9):: CALL HCHAR(
15,3,65,9):: CALL HCHAR(16,8,42,9):: CAL
L HCHAR(17,8,65,9):: CALL HCHAR(17,18,64
,15)
1840 CALL SPRITE(#3,124,4,150,90,#5,124,
4,150,187,#4,108,16,142,240,0,-7)
1850 CALL SPRITE(#1,136,16,144,5,#2,132,
5,144,5)
1860 SUBEND
1870 SUB EIGHT
1880 CALL DELSPRITE(#1,#2,#3,#4,#5)
1890 CALL HCHAR(9,1,33,352)
1900 DISPLAY AT(11,1):"!!!!!!!!**!!!*!!!
******!!!!!!*!!!*!!AA!*!A!!*AAAAAA!**!!!
A!*!A*!!!*A!!!A!!*****AA!*!!!A!!A!*!A!!
*!*!**AAAAA!!!A!!!!!!!!A!!!!A!A!AA!!!!!
!!!!";
1910 CALL HCHAR(10,1,33,32)
1920 CALL HCHAR(17,1,64,15)
1930 CALL SPRITE(#3,128,2,144,80,#4,108,
16,142,112,#5,124,4,150,152)
1940 CALL SPRITE(#1,136,16,144,5,#2,132,
5,144,5)
1950 SUBEND
1960 SUB NINE
1970 CALL DELSPRITE(#1,#2,#3,#4,#5)
1980 CALL HCHAR(9,1,33,352)
1990 DISPLAY AT(11,1):"!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*!!!!!
!!!!!!!!!!*!!!!!!!!!!!A!!!!!!!!!!!!!!!A!
!!!!!!!!!!!!!!!!";
2000 CALL HCHAR(10,1,33,32)
2010 CALL HCHAR(17,1,64,32)
2020 CALL SPRITE(#3,128,2,144,80,#5,124,
4,150,112,#4,108,16,142,240,0,-7)
2030 CALL SPRITE(#1,136,16,144,5,#2,132,
5,144,5)
2040 SUBEND
2050 SUB SPRUNG(X,PKT,LIVE)
2060 CALL COINCI(PKT,LIVE)
2070 FOR I=-20 TO 0 STEP 4 :: CALL MOTIO
N(#2,I,((X*X)/4)*2.3,#1,I,((X*X)/4)*2.3)
:: NEXT I
2080 FOR I=0 TO 20 STEP 4 :: CALL MOTION
```

```
(#2,I,((X*X)/4)*2,#1,I,((X*X)/4)*2):: NE
XT I
2090 CALL MOTION(#2,0,0,#1,0,0)
2100 CALL POSITION(#2,A,B)
2110 CALL LOCATE(#1,144,B,#2,144,B)
2120 CALL COINCI(PKT,LIVE)
2130 CALL SOUND(50,-1,0)
2140 SUBEND
2150 SUB COINCI(PKT,LIVE)
2160 CALL COINC(#1,#3,10,Z1)
2170 CALL COINC(#1,#4,9,Z2)
2180 CALL COINC(#1,#5,10,Z3)
2190 IF (Z1+Z2+Z3)<>0 THEN 2200 ELSE SUB
EXIT
2200 IF Z2<>0 THEN LIVE=LIVE-1 ELSE 2240
2210 IF LIVE=0 THEN CALL ENDE
2220 CALL DELSPRITE(#(10-LIVE)):: CALL V
CHAR(22,15-LIVE*3,41,2):: CALL VCHAR(22,
16-LIVE*3,41,2)
2230 FOR I=0 TO 30 STEP 2 :: CALL SOUND(
-555,-7,I):: NEXT I :: CALL LOCATE(#1,14
4,5,#2,144,5):: SUBEXIT
2240 PKT=PKT-20 :: CALL SOUND(-100,-7,0)
:: CALL BUMM
2250 DISPLAY AT(1,10)SIZE(6):PKT :: CALL
 LOCATE(#1,144,5,#2,144,5)
2260 SUBEND
2270 SUB NEUBILD(BILD(),PKT,BONI)
2280 IF BIL=6 THEN IF PKT>BONI THEN CALL
 BONUS(PKT)
2290 BIL=BIL+1
2300 IF BIL=10 THEN CALL NEU(BILD()):: B
IL=0 :: BONI=BONI+1950 :: GOTO 2290
2310 ON BILD(BIL)GOSUB 2330,2340,2350,23
60,2370,2380,2390,2400,2410
2320 SUBEXIT
2330 CALL ONE :: RETURN
2340 CALL TWO :: RETURN
2350 CALL THREE :: RETURN
2360 CALL FOUR :: RETURN
2370 CALL FIVE :: RETURN
2380 CALL SIX :: RETURN
2390 CALL SEVEN :: RETURN
2400 CALL EIGHT :: RETURN
2410 CALL NINE :: RETURN
2420 SUBEND
2430 SUB BUMM
2440 CALL MOTION(#1,0,0,#2,0,0)
2450 CALL MOTION(#1,1,0,#2,1,0)
2460 FOR I=1 TO 250 :: NEXT I
2470 CALL MOTION(#1,0,0,#2,0,0)
2480 FOR I=1 TO 100 :: NEXT I
2490 SUBEND
2500 SUB NEU(BILD())
2510 FOR I=1 TO 9 :: BILD(I)=0 :: NEXT I
2520 FOR I=2 TO 9
2530 ZUFA=INT(RND*9)+1
2540 IF BILD(ZUFA)<>0 THEN 2530
2550 BILD(ZUFA)=I
2560 NEXT I
2570 FOR I=1 TO 9 :: IF BILD(I)=0 THEN B
ILD(I)=2 :: GOTO 2590
2580 NEXT I
2590 SUBEND
2600 SUB ENDE
2610 CALL MOTION(#1,0,0,#2,0,0)
2620 FOR I=0 TO 30 :: CALL SOUND(-50,550
,I):: NEXT I
2630 CALL COLOR(#2,6):: CALL KEY(2,K,S):
: IF S=0 THEN CALL COLOR(#2,9):: GOTO 26
30 ELSE CALL DELSPRITE(ALL):: RUN
2640 SUBEND
```

„Mayday! - Mayday!" - Feindliche Ufos im Anflug - „Mayday! - Mayday!" Feindliche . . .

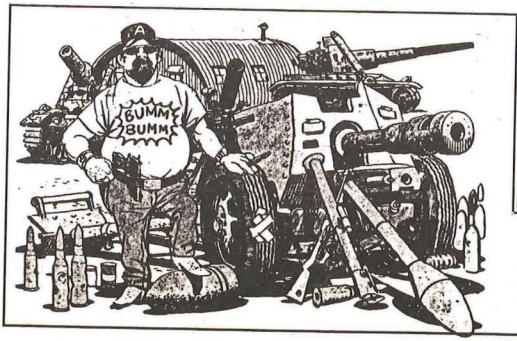
Desert Attack

TI-99/4 A (Extended Basic)

Nach Tagen und Wochen des intensiven und störungsfreien Arbeitens wird Ihre Forschungsstation plötzlich von außerirdischen Flugobjekten angegriffen. Ihnen bleibt nichts anderes übrig, als mit aller Macht zu versuchen, die Angreifer aufzuhalten. Zur Ausführung dieses Vorhabens befinden Sie sich im Besitz einer Ufo-Abwehr-Kanone.

Nachdem es Ihnen gelungen ist, vier Angriffswellen, bestehend aus jeweils 10 Ufos, abzuwehren, bricht die Nacht heran und Sie glauben, das Schlimmste sei überstanden. Ihrer Hoffnung auf eine ruhige Nacht setzt das plötzliche Auftauchen feindlicher Panzer jedoch ein jähes Ende. Nun heißt es erneut, Ihre Abwehrwaffe sicher und zielgenau gegen die Angreifer einzusetzen.

Als junger und noch unerfahrener Wissenschaftler, der inder Welt der Forschung noch keinen Namen hat, erhalten Sie überraschenderweise einen Forschungsauftrag Ihrer Regierung. Mit jugendlicher Begeisterung nehmen Sie diesen Auftrag, der Sie zu einer Forschungsstation in der Wüste Sagobi führt, ohne viel nachzudenken, an. Dort angekommen, warten jedoch einige böse Überraschungen auf Sie. (S.G.)



Action u. tolle Grafik

Unser „Held" scheint auf den Angriff der Außerirdischen gut vorbereitet zu sein

DESERT ATTACK – DAS LISTING

```
10 GOSUB 1330
20 RANDOMIZE
30 CALL CHAR(140,"0101010101010101010101
0101010101808080808080808080808080808080
80")
40 MS1$="0001000100010001000100010001000
1008000800080008000800080008000800080"
50 MS2$="0100010001000100010001000100010
0800080008000800080008000800080008"
60 !
70 REM *** MONSTERCHAR ***
80 !
90 C$(1)="0000000103040F7FFF7F0C04000000
0000000080C020F0FEFFFE3020"
100 C$(2)="0010387CEDC782040703060C18240
00000081C3EB7E34120E0C060301824"
110 C$(3)="00000008140C070D3F0C030408300
80000000081418F0B0FC30C020100C10"
120 C$(4)="040202091B3F7FF1617F040810103
87C20404090A8FCFE8F86FC201008081C3E"
130 C$(5)="000000100804060F1D3767C382040
0000000008102060F0B8ECE6C34120"
140 C$(6)="00804123120F7FCD7F3F000000000
000000182C448E0FEB3FEFC"
150 C$(7)="0000000001073F27271C07043C545
2000000000080E0F8E4E438E0203C2A4A"
160 C$(8)="002422E1110987453F44870911E12
2240048880E1020C244F844C220100E8848"
170 !
180 REM ** EXPLOSIONCHAR ***
190 !
200 EX$(1)="00410A0020010A01044804200400
8000020048020401200508002880240001"
210 EX$(2)="0000000000010A01040804000000
0000000008020401000508002080"
220 EX$(3)="0000000000010201040000000000
0000000000000000000004080"
230 P$="000000080F00000000007FAA7F000000
0000003CCE3E3C3878FCFEABFE" !*** PANZER
***
240 B$="807F201F08070201010001000100101
01FE04F810E0408080800080008000000" !* BOM
BE *
250 M$="030F1E3C7C4DEFFFFCFEF8E37F3F1F03
E08000000080800000002061EFEFCF8F0" !** MO
ND **
260 CALL CHAR(132,"00010608102020407F402
0201008060100C0B08884828281FF8182828488B
0C0")!*** FADENKREUZ ***
270 S$="030F1F3F7F7FFFFFFFFF7F7F3F1F0F03
C0F0F8FCFEFEFFFFFFFFFEFEFCF8F0C0" :: CAL
L CHAR(136,S$)!*** SONNE ***
280 !
290 REM *** STATIONCHAR UND
AUFBAUTEN ! ***
300 !
310 CALL CHAR(112,"0102060A1A286BA6",113
,"FFAAAA00FFFF",114,"AC98B060C080",115,"
FFAAFFFFAA")
320 CALL CHAR(97,"0103070F1F3F7FFF",98,"
8181C3E7FFFFFFFF",99,"00000000183C7EFF",
100,"80C0E0F0F8FCFEFF",101,RPT$("FF",8))
330 CALL CLEAR :: CALL CHAR(104,RPT$("FF
",8)):: CALL HCHAR(9,1,104,512)
340 CALL COLOR(9,5,8,1,1,8,10,9,1,11,2,9
,12,16,9)
350 FOR I=2 TO 8 :: CALL COLOR(I,13,8)::
 NEXT I
360 DISPLAY AT(4,1):" c    c        c
  cc     caed aed      aed      aeed  aeeee
beeed    aeeed    aeeeeebeeeeeeeeed aeeee
ebbbeeeeeeeeeeeeeeeebeeeeeeeeeeeeeeeeee
e"
370 CALL HCHAR(6,2,97):: CALL VCHAR(7,1,
101,2):: CALL VCHAR(7,2,101,2)
380 CALL HCHAR(5,31,100):: CALL HCHAR(6,
32,100):: CALL VCHAR(6,31,101,3):: CALL
VCHAR(7,32,101,2)
390 CALL HCHAR(16,8,113,12):: CALL HCHAR
(16,7,112):: CALL HCHAR(17,7,114):: CALL
 HCHAR(17,6,112):: CALL HCHAR(18,6,114)
400 CALL HCHAR(18,5,112):: CALL HCHAR(19
,5,114):: CALL HCHAR(19,4,112):: CALL HC
HAR(20,4,115,17)
410 CALL HCHAR(19,19,120,2):: CALL HCHAR
(17,18,120):: CALL HCHAR(18,20,120):: CA
LL HCHAR(16,19,120)
420 CALL HCHAR(18,9,121):: CALL HCHAR(18
,19,120):: CALL HCHAR(18,16,121):: CALL
HCHAR(17,14,121)
430 CALL VCHAR(17,22,122,2):: CALL VCHAR
(17,23,122,2)
440 CALL CHAR(120,"00FF99FFFF99FFFF",121
,"001038787808081C",122,"0040F8500040F85
0")
450 FOR I=2 TO 8 :: CALL COLOR(I,13,8)::
 NEXT I :: W,SP,PK,CH=0 :: CALL SCREEN(1
3):: CALL MAGNIFY(3):: CALL SPRITE(#28,1
36,12,12,80)
460 CALL SPRITE(#1,132,15,100,128)
470 GOSUB 800 :: GOSUB 610
480 L$=C$(CH)
490 CALL MOTION(#1,0,0):: DISPLAY AT(1,1
):"PUNKTE:";PU :: AN=AN+1 :: IF AN=11 TH
EN AN=0 :: GOTO 460
500 IF W=5 AND ZS=0 THEN GOSUB 690
510 IF W=9 THEN GOSUB 750 ELSE IF W=6 AN
D ZT=0 THEN 910
520 FOR I=30 TO 0 STEP -1 :: CALL SOUND(
-200,-5,I,900,I,700,I):: NEXT I
530 F=INT(13*RND)+4 :: IF F=9 THEN 530 E
LSE CALL COLOR(#2,F):: CALL SPRITE(#2,12
8,F,1,INT(180*RND)+40)
540 CALL JOYST(1,X,Y):: CALL KEY(1,K,S)
: CALL MOTION(#1,-Y*4,X*4):: CALL POSITI
ON(#2,P,Z)
550 IF P>=140 THEN 1120
560 IF K=18 THEN CALL SOUND(50,-7,0):: C
ALL COINC(#1,#2,12,C):: IF C THEN GOSUB
1070 :: PU=PU+PK :: GOTO 490
570 CALL MOTION(#2,SP,INT(40*RND)-20)::
GOTO 540
580 !
590 REM ** WELCHES MONSTER              K
OMMT JETZT ? **
600 !
610 ZT,ZS=0 :: CALL DELSPRITE(#2):: W=W+
1 :: CH=CH+1 :: PK=PK+100
620 IF CH=9 THEN CH=1
630 CALL CHAR(128,C$(CH))
640 SP=SP+2
650 RETURN
660 !
```

```
670 REM ** ES WIRD NACHT **
680 !
690 CALL SOUND(200,800,0,820,0):: CALL S
OUND(200,800,0,830,0):: CALL SOUND(200,8
00,0,840,0):: CALL SOUND(600,1200,0,1220
,0,1130,0)
700 ZS=1 :: CALL SCREEN(14):: CALL COLOR
(1,1,2,9,5,2):: CALL CHAR(136,M$)
710 RETURN
720 !
730 REM ** ES WIRD TAG **
740 !
750 CALL SCREEN(13):: CALL COLOR(1,1,8,9
,5,8):: CALL CHAR(136,S$)
760 W=0 :: RETURN
770 !
780 REM ** BOMBENANGRIFF **
790 !
800 FOR D=1 TO 4 :: FOR I=110 TO 300 STE
P 10 :: CALL SOUND(-200,I,0,I+20,0):: NE
XT I :: NEXT D
810 CALL CHAR(128,B$):: L$=B$
820 R=R+1 :: IF R=7 THEN R=0 :: RETURN
830 F=INT(13*RND)+4 :: IF F=9 THEN 820 E
LSE CALL SPRITE(#2,128,F,1,INT(200*RND)+
20,INT(20*RND)+1,0)
840 CALL JOYST(1,X,Y):: CALL KEY(1,K,S):
: CALL MOTION(#1,-Y*4,X*4):: CALL POSITI
ON(#2,P,Z)
850 IF P>=140 THEN 1120
860 IF K=18 THEN CALL SOUND(50,-7,0):: C
ALL COINC(#1,#2,10,C):: IF C THEN GOSUB
1070 :: GOTO 820
870 GOTO 840
880 !
890 REM *** BONUSRUNDE ***
900 !
910 ZT=1 :: CALL DELSPRITE(#2):: CALL CH
AR(128,P$):: L$=P$
920 FOR D=1 TO 4
930 FOR I=0 TO 30 STEP 5 :: DISPLAY AT(1
2,9):"BONUSRUNDEhhhhhhhhhh" :: CALL SOUN
D(-200,110,I):: NEXT I
940 FOR V=1 TO 70 :: NEXT V
950 DISPLAY AT(12,1):"hhhhhhhhhhhhhhhhhh
hhhhhhhhh"
960 FOR V=1 TO 70 :: NEXT V :: NEXT D
970 ZH=0
980 DISPLAY AT(1,1):"PUNKTE:";PU :: ZH=Z
H+1 :: IF ZH=9 THEN CALL MOTION(#1,0,0):
: GOTO 460
990 F=INT(13*RND)+4 :: IF F=9 THEN 990 E
LSE CALL SPRITE(#2,128,F,INT(120*RND)+70
,250,0,-INT(30*RND)-8)
1000 CALL JOYST(1,X,Y):: CALL KEY(1,K,S)
:: CALL MOTION(#1,-Y*4,X*4):: CALL POSIT
ION(#2,P,Z)
1010 IF K=18 THEN CALL SOUND(50,-5,0):: 
CALL COINC(#1,#2,10,C):: IF C THEN GOSUB
 1070 :: PU=PU+100 :: GOTO 980
1020 IF Z<=10 THEN CALL DELSPRITE(#2):: 
GOTO 980
1030 GOTO 1000
1040 !
1050 REM *** ABSCHUSS ***
1060 !
1070 CALL MOTION(#1,0,0):: CALL SOUND(-7
00,-6,0,140,0):: FOR I=1 TO 3 :: CALL CH
AR(128,EX$(I)):: NEXT I :: CALL DELSPRIT
E(#2)
1080 CALL CHAR(128,L$):: RETURN
1090 !
1100 REM *** UFOLANDUNG ***
1110 !
1120 CALL MOTION(#1,0,0):: FOR I=1 TO 38
 :: CALL SOUND(-100,-7,9,900,9):: CALL M
OTION(#2,2,0):: NEXT I
1130 CALL MOTION(#2,0,-6)
1140 CALL SOUND(-200,110,0,130,0,170,0):
: CALL POSITION(#2,P1,Z1):: IF Z1=143 OR
 Z1=144 THEN 1150 ELSE 1140
1150 DISPLAY AT(1,1):"" :: CALL MOTION(#
2,0,0):: CALL SPRITE(#3,140,5,158,146)
1160 FOR I=1 TO 50 :: CALL SOUND(-200,40
0,0,410,0):: CALL COLOR(#3,INT(13*RND)+4
):: NEXT I
1170 CALL SOUND(400,-7,0,110,0,110,0):: 
CALL CHAR(120,"0008387DFF99FFFF")
1180 CALL CHAR(120,"0000000008387CFF")
1190 CALL DELSPRITE(ALL):: DISPLAY AT(14
,6):"NEUEShSPIELh?hJ-Nhhhhhh"
1200 IF PU>APU THEN DISPLAY AT(23,1):"NE
UERhHISCORE:";PU;"hhhhhhhh" :: APU=PU ::
 PU=0
1210 FOR I=2 TO 8 :: CALL COLOR(I,13,9):
: NEXT I
1220 DISPLAY AT(10,10):"GAMEhOVERhhhhhhh
hhh"
1230 FOR D=1 TO 100 :: NEXT D
1240 DISPLAY AT(10,1):"hhhhhhhhhhhhhhhhh
hhhhhhhhhh"
1250 FOR D=1 TO 100 :: NEXT D
1260 CALL KEY(0,K,S):: IF S=0 THEN 1220
ELSE IF K=74 OR K=106 THEN 1280 ELSE IF
K=78 OR K=110 THEN CALL CLEAR :: END
1270 GOTO 1260
1280 DISPLAY AT(12,1):"hhhhhhhhhhhhhhhhh
hhhhhhhhhh" :: DISPLAY AT(14,1):"hhhhhh
hhhhhhhhhhhhhhhhhhh"
1290 DISPLAY AT(23,1):"hhhhhhhhhhhhhhhhh
hhhhhhhhhh" :: GOTO 440
1300 !
1310 REM ** VORSPANN **
1320 !
1330 CALL CLEAR :: CALL SCREEN(2):: FOR
I=2 TO 8 :: CALL COLOR(I,12,1):: NEXT I
1340 T$(1)="          DESERT  ATTACK"
1350 T$(2)="     (C) BY HORST SPIERLING"
1360 T$(3)="          JUNI    1984"
1370 CALL CHAR(42,"13D67E3CFC3E7A91"):: 
```

Noch wenige Zeilen zum Eintippen und dann beginnt für Sie die gefährliche Forschungsarbeit in der Wüste Sagobi

```
CALL VCHAR(11,1,42,3):: CALL VCHAR(11,32
,42,3)
1380 RD=1 :: CALL KEY(1,K,S):: IF K=18 T
HEN RETURN
1390 FOR I=3 TO LEN(T$(RD)):: IF ASC(SEG
$(T$(RD),I,I))=32 THEN 1410 ELSE CALL HC
HAR(RD+10,I,42)
1400 CALL KEY(1,K,S):: IF K=18 THEN RETU
RN ELSE CALL HCHAR(RD+10,I,ASC(SEG$(T$(R
D),I,I)))
1410 FOR D=1 TO 30 :: NEXT D :: NEXT I
1420 RD=RD+1 :: IF RD<4 THEN 1390
1430 CALL KEY(1,K,S):: IF K=18 THEN RETU
RN ELSE RD=1
1440 FOR D=1 TO 200 :: NEXT D
1450 FOR I=3 TO 32 :: CALL HCHAR(RD+10,I
,42):: CALL HCHAR(RD+10,I-1,32):: NEXT I
1460 CALL KEY(1,K,S):: IF K=18 THEN RETU
RN ELSE RD=RD+1 :: IF RD<4 THEN 1450 ELS
E 1380
```

DESERT ATTACK — *The End*

★ CAR CRASH ★ CAR CRASH ★ CAR CRASH ★

Ein schnelles Labyrinthspiel für den ZX-Spectrum mit 16/48 K

Sie brettern mit Ihrem Turbo-Buggy durch zehn verschiedene Bildschirme, wobei Sie in jedem Bildschirm sechs Benzinkanister aufsammeln müssen – klar, ohne Treibstoff läuft gar nichts. Wenn Sie dies getan haben, dann fahren Sie zum einzigen Ausgang, und mit dem nächsten Labyrinth geht's weiter. Ihr Ziel ist der goldene Pokal für den besten Buggy-Driver. Aber Sie sind nicht der einzige, der die begehrte Trophäe gewinnen will – vier giftgrüne Rammbuggys behindern Sie bei Ihrer Jagd nach dem Pokal. Sie können ihnen ausweichen, indem Sie wegfahren (Vorsicht: die Gegner sind genauso schnell wie Sie) oder mit Ihren Bordraketen auf den Gegner feuern. Allerdings sollte man sich nicht in jedem Fall auf seine eigene Feuerkraft verlassen; es gibt nämlich Superbuggys, die erst durch mehrere Treffer zerstört werden können...

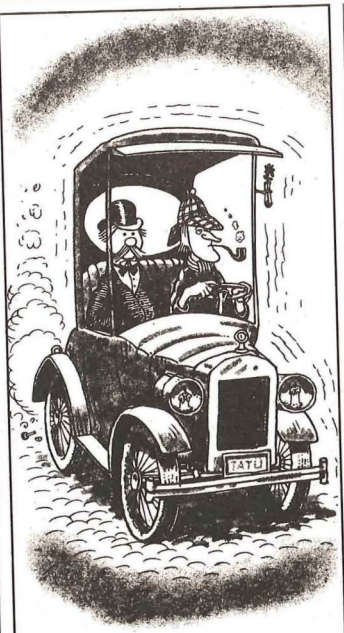
Sie können Ihren Buggy mit den Tasten 6, 7, 8 und 9 steuern und mit 0 Ihre Raketen abfeuern. Wer einen Kempston-Joystick hat, kann auch den benutzen. Noch ein Tip: Um die Bordraketen abfeuern zu können, müssen Sie Ihren Buggy kurz anhalten – erstens aufgrund des enormen Rückstoßes und zweitens, weil Sie sonst der Feuerstrahl verbrennt. Außerdem sollten Sie stets darauf achten, daß Sie der Gegner nicht in der Ausgangsposition erwischt, sonst ist das Spiel für Sie gelaufen.

Eingabehinweise

Das Programm besteht aus drei Listings: Listing 1 ist das Basic-Programm, das das Menue und die Anfangsmelodie enthält und den Maschinencode aufruft. Damit das Spiel schnell genug wird, hat der Autor es vollständig in Maschinensprache geschrieben. Geben Sie also zunächst Listing 1 ein und speichern Sie es mit „GOTO 9999" auf Kassette ab. Sie dürfen es aber auf keinen Fall starten. Löschen Sie mit „NEW", nachdem Sie es abgespeichert haben und geben Sie nun Listing 2 ein. Dieses Listing starten Sie bitte mit „RUN". Es enthält den Maschinencode und liest diesen in den Speicher.

Wenn Sie keinen Fehler begangen haben, dann wird der Code automatisch hinter dem Basic-Listing abgespeichert. Andernfalls müssen Sie die betreffende Programmzeile noch einmal überprüfen. Verfahren Sie genauso mit dem Listing 3. Wenn der Code abgespeichert ist, können Sie Listing 2 löschen und Listing 3 eingeben. Es enthält die Bildschirme und einen zweiten Zeichensatz. Das Basic-Listing schaltet automatisch auf den zweiten Zeichensatz um. Sie können wieder auf den alten umschalten, wenn Sie „POKE 23607, 60" eingeben.

Wenn Sie das gesamte Programm eingegeben und abgespeichert haben, dann können Sie dieses komplett von Kassette laden.



Es ist nicht zu glauben, aber diese beiden Herren wollen in diesem Gefährt tatsächlich an „Car-Crash" teilnehmen.

Stellen Sie Ihr Reaktionsvermögen unter Beweis

Erkämpfen Sie sich den Goldpokal und lassen Sie sich zum besten Buggydriver küren!

Ein außerordentlich ideenreiches Spiel für den TI 99/4A mit Extended Basic und Joysticksteuerung

SCOOTIE

Spielaufgabe ★ Spielaufgabe ★ Spielaufgabe ★ Spielaufgabe

Es geht um Scootie, den glücklichen Hausbesitzer. Er hat beschlossen, sein Dach zu isolieren. Zu diesem Zweck hat er Isolationsmaterial im Keller gestapelt und will es nun nach oben tragen und am Dach anbringen (mit Feuerknopf). Seine Kinder rollen jedoch ständig irgendwelche Gegenstände auf den Flur, über die Scootie hinwegspringen muß. Außerdem leidet Scootie an einer Magen-Darm-Grippe, und muß ständig (jedes Mal wenn im Programm der Schriftzug „Toilette" auftaucht) das „Örtchen" aufsuchen. Zu allem Unglück ereignete sich im Keller auch noch ein Wasserrohrbruch, durch den das Isolatiosmaterial zerstört würde. Deshalb muß Scootie nun auch Sandsäcke aus dem ersten Stock in den Keller tragen, um das Material zu schützen. Stürzt unser Held beim Transport, so wird das Material beim Herunterfallen zerstört. Damit das Spiel weitergehen kann, muß Scootie erst die Überreste mit dem Eimer zur Mülltonne transportieren.

Eingabetips ★ Eingabetips ★ Eingabetips ★ Eingabetips

Zuerst muß das Programm „Scootie-Grafik" abgetippt werden. Dieses wird dann mit „RUN" gestartet. Die vom Programm generierte Grafik-Datei muß nun auf Band aufgenommen werden. Anschließend kann man „Scootie-Grafik" löschen.

Jetzt ist das Hauptprogramm einzugeben. Dabei sollte man sehr sorgfältig vorgehen, da später nach jeder Fehlermeldung die Grafik zurückgesetzt wird. Dann kann das Hauptprogramm gestartet werden. Anschließend muß die Grafik-Datei geladen werden. Achtung! Nachdem auf die Anweisung „PRESS CASSETTE START, THAN PRESS ENTER" hin ENTER gedrückt wurde, darf die Grafik-Datei erst dann geladen werden, wenn der Bildschirm um eine Zeile nach oben gerückt ist! Jetzt kann das Spiel begonnen werden. Das Spiel ist zu Ende, wenn der Wasserspiegel den Sandsackhaufen übersteigt, Scootie mehr als fünfmal abgestürzt oder das Dach komplett isoliert ist. Die Alpha-Lock-Taste muß natürlich ausgerastet sein.

Und nun zum Listing

```
100 CALL MAGNIFY(3):: RANDOMIZE
110 DIM A$(4),AM$(7),C(1),XS(7),YS(7),EX
(5),SA(1),S(2)
120 FOR I=0 TO 7 :: READ XS(I),YS(I):: N
EXT I
130 FOR I=0 TO 5 :: READ EX(I):: NEXT I
140 C(0)=57 :: C(1)=33 :: SA$="000000007
EFFFF7E"
200 OPEN #1:"CS1",INPUT ,SEQUENTIAL,INTE
RNAL,FIXED 65
210 FOR I=0 TO 7 :: INPUT #1:AM$(I):: NE
XT I
220 FOR I=0 TO 4 :: INPUT #1:A$(I):: NEX
T I
230 FOR I=32 TO 56 STEP 4 :: INPUT #1:B$
 :: CALL CHAR(I,B$):: NEXT I
240 FOR I=64 TO 116 STEP 4 :: INPUT #1:B
$ :: CALL CHAR(I,B$):: NEXT I
250 CLOSE #1
300 CALL COLOR(2,2,1):: DISPLAY AT(1,12)
ERASE ALL:"SCOOTIE":TAB(12);"-------"
310 FOR I=0 TO 4 :: CALL CHAR(120+4*I,A$
(I)):: DISPLAY AT(6+2*I,2):I+1;"FOR" ::
CALL SPRITE(#I+1,120+4*I,7,33+16*I,81)::
 NEXT I
320 DISPLAY AT(16,3):"6 FOR LEICHT" :: D
ISPLAY AT(18,3):"7 FOR SCHWER" :: DISPLA
Y AT(21,5):"BITTE WJHLEN"
```

Teil Eins

```
330 HI=MAX(INT(P/14.4),HI):: DISPLAY AT(
24,1):"BESTLEISTUNG,";HI;"/"
340 CALL KEY(0,RET,STA):: IF RET>55 OR R
ET<49 THEN 340 ELSE SW=RET-49 :: CALL DE
LSPRITE(ALL):: CALL CLEAR
400 DEF HIND(T)=-SW*(SW<5)+INT(RND*T/250
)*(SW=5)*(T<1250)+ABS(INT(RND*5)*((SW=6)
OR((T>1250)AND(SW>4))))
410 S(0),S(1)=3 :: S(2)=2
420 SA(0),SA(1)=1 :: LE=5 :: T,HW,P,PL,F
,SN,K,K1=0
430 CALL CHAR(120,"0103030301030708080B0
B063E3C20008040C0C080C0E4E8D0C0C0C06060C
0E")
440 CALL CHAR(124,"0102030301032717080B0
3030606030780C0C0C080C0E0D0D0D0D0607C3C0
4")
450 CALL CHAR(39,"0",107,"0",108,A$(HIND
(T)),136,A$(HIND(T)),140,A$(HIND(T)),128
,AM$(4),132,AM$(5))
460 FOR I=1 TO 11 :: CALL COLOR(I,1,1)::
 NEXT I
470 FOR I=0 TO 4 :: CALL HCHAR(13+I,25+I
,89):: CALL HCHAR(18+I,10-I,88):: NEXT I
480 FOR I=0 TO 5 STEP 5 :: CALL HCHAR(12
+I,1,112):: CALL HCHAR(12+I,2,113):: CAL
L HCHAR(11+I,1,116):: CALL HCHAR(11+I,2,
118):: NEXT I
```

```
490 CALL TUER(57,81,1):: CALL TUER(33,12
1,0):: CALL TUER(233,81,1)
500 RESTORE 2610 :: FOR I=1 TO 22 :: REA
D Y3,X3,A,B :: CALL HCHAR(Y3,X3,A,B):: N
EXT I
510 CALL VCHAR(8,10,92,5):: CALL VCHAR(8
,11,93,5)
520 FOR I=1 TO 24
530 Y=INT(RND*3)+3 :: X=INT(RND*28)+3 ::
 CALL GCHAR(Y,X,A):: IF A=40 THEN 530
540 CALL HCHAR(Y,X,40):: NEXT I
550 DISPLAY AT(1,1):"PUNKTE: 0";TAB(17);
"hhhhh"
560 CALL COLOR(1,7,1,2,1,13,3,2,1,4,2,1,
5,2,1,6,2,1,7,2,1,8,2,1,9,9,2,10,6,1,11,
3,1)
570 CALL SPRITE(#15,120,5,41,128,#16,36,
1,25,132)
580 FOR I=1 TO 3 :: READ A,B :: CALL SPR
ITE(#I,A,B,81,57,0,-12):: FOR N=1 TO 400
 :: NEXT N :: NEXT I
590 CALL TUER(57,81,0)
700 E=-2
710 CALL JOYST(1,X,Y):: CALL PATTERN(#15
,122-X/2):: CALL MOTION(#15,0,X*2,#16,0,
X*2)
720 T=T+.3 :: IF X=0 THEN CALL KEY(1,RET
,STA):: IF RET=18 THEN ON E+4+SN*2 GOTO
1100,800,1200
730 IF F+4*(E+2)=0 THEN CALL COINC(#15,-
(E+2)*120+41,XS((E+3)*2),5,A):: IF A AND
((Y=-4)+(E=-3))THEN 750
740 GOTO 710
750 CALL MOTION(#15,0,0,#16,0,0):: IF E=
-3 THEN 2210
760 V=-8 :: CALL MOTION(#1,0,V,#2,0,V,#3
,0,V):: GOTO 1010
800 FOR I=-8 TO 8 :: CALL MOTION(#15,I,0
,#16,I,0):: CALL SOUND(-200,600-ABS(I)*1
0,0)
810 CALL KEY(1,RET,STA):: IF RET<>18 OR
F=0 THEN 870
820 T=T+.7 :: CALL MOTION(#15,0,0,#16,0,
0):: CALL POSITION(#16,Y1,X1):: IF X1>24
0 THEN 870
830 X3=X1/8+1 :: Y3=Y1/8+2.5
840 CALL GCHAR(Y3,X3,Z):: IF Z<>34 THEN
870
850 CALL HCHAR(Y3,X3,40):: P=P+20 :: DIS
PLAY AT(1,8)SIZE(5):P
860 F=F-1 :: PL=PL+1 :: CALL CHAR(37,RPT
$("00",8-F)&RPT$("FF",F))
870 NEXT I
880 T=T+4.2 :: CALL MOTION(#15,0,0,#16,0
,0)
890 CALL POSITION(#15,Y1,X1):: CALL LOCA
TE(#15,41,X1,#16,25,X1+4)
900 IF PL<60 THEN 710
910 CALL DELSPRITE(ALL):: CALL COLOR(2,2
,1):: DISPLAY AT(10,13)ERASE ALL:"BRAVO"
920 DISPLAY AT(13,1):"SIE HABEN ALLE PLA
TTEN ANGE-BRACHT." :: DISPLAY AT(15,1):U
SING "SIE BENMTIGTEN #### SEK.":T
930 FOR G=1 TO 10 :: FOR N=20 TO 1 STEP
-1 :: CALL SOUND(-10,2000+10*N*N,0):: NE
XT N :: NEXT G
940 GOTO 300
1000 CALL CHAR(128,AM$(4),132,AM$(5))
1010 CALL POSITION(#15,Y1,X1):: N=128
1020 CALL JOYST(1,X,Y):: IF Y=0 THEN 102
0
```

„Herr Müller, falls Sie Ihre Eintippbemühungen derart ermatten, daß Ihre Arbeitsleistung darunter leidet, dann möchte ich Ihnen einen Tip geben. Machen Sie's so wie ich und bestellen Sie die tollen TRONIC-Programme einfach über den Software-Katalog!"

```
1030 T=T+.2 :: Y1=Y1-Y
1040 N=260-N :: CALL SOUND(-30,1000-5*Y1
,0):: CALL PATTERN(#15,N):: CALL LOCATE(
#15,Y1,73,#16,Y1-16,77)
1050 ON INT((Y1-42)/39)+2 GOTO 700,1020,
1300
1100 CALL POSITION(#15,Y1,X1):: IF X1>10
4 OR X1<64 THEN 710
1110 CALL CHAR(37,"00000000FFFFFFFF"):: 
CALL SPRITE(#16,36,3,Y1-16,X1+4)
1120 FOR G=1 TO 6 :: CALL SOUND(1,523,0)
:: CALL SOUND(1,698,0):: NEXT G
1130 F=4 :: T=T+1.32 :: GOTO 710
1200 CALL POSITION(#16,Y2,X2):: IF X2<14
1 OR X2>155 THEN 710
```

```
1210 X3=1+(SA(0)<SA(1)):: SA(X3)=SA(X3)+
1
1220 CALL COLOR(#16,1):: CALL HCHAR(22-I
NT((SA(X3)-.5)/2),X3+19,34-(-1)^SA(X3))
1230 CALL SOUND(10,523,0):: FOR G=0 TO 3
0 STEP 2 :: CALL SOUND(-10,2794-G*50,G):
: NEXT G
1240 P=P+30 :: DISPLAY AT(1,8)SIZE(5):P
:: SN=0 :: T=T+1.9 :: GOTO 710
1300 FOR I=1 TO 7 :: CALL JOYST(1,X,Y)::
 CALL PATTERN(#15,122-X/2):: CALL MOTION
(#15,0,X*2,#16,0,X*2)
1310 T=T+.44 :: CALL COINC(#15,YS(E+3),X
S(E+3),5,A):: IF A THEN 1380
1320 CALL KEY(1,RET,STA):: IF E<1 THEN C
ALL POSITION(#15,Y1,X1):: IF X1<49 AND X
+RET+SN=18 THEN 2100
1330 IF RET=18 THEN 1400
1340 CALL COINC(ALL,A):: CALL COINC(#15,
YS(E+3),1,4,B):: IF A OR B THEN 1500
1350 NEXT I
1360 IF INT(RND*10)=9 THEN 1900
1370 GOTO 1300
1380 CALL MOTION(#15,0,0,#16,0,0):: ON I
NT(LOG((ABS(E+.75)+.25)*2)+1.99)GOTO 100
0,1350,2200,1940
1400 CALL MOTION(#16,-18,X*1.25,#15,-18,
X*1.25):: CALL SOUND(20,20000,0):: CALL
COINC(ALL,A):: IF A THEN 1500
1410 FOR N=-14 TO 17.5 STEP 3.5 :: CALL
MOTION(#15,N,X*1.25,#16,N-SGN(N),X*1.25)
:: CALL COINC(ALL,A):: IF A THEN 1500
1420 NEXT N :: CALL SOUND(10,-5,0)
1430 CALL MOTION(#15,0,0,#16,0,0):: CALL
 POSITION(#15,Y1,X1):: CALL LOCATE(#15,Y
S(E+3),X1,#16,YS(E+3)-16,X1+4)
1440 T=T+1.9 :: GOTO 1350
1500 CALL MOTION(#15,0,0,#16,0,0):: CALL
 CHAR(128,AM$(INT(6.5-X/8))):: CALL PATT
ERN(#15,128)
1510 CALL POSITION(#15,Y1,X1):: CALL LOC
ATE(#15,Y1-6,X1):: CALL MOTION(#15,4,X,#
16,4,1.5*X)
1520 IF SN+F=0 THEN 1550
1530 CALL DISTANCE(#15,#1,X1):: CALL DIS
TANCE(#15,#2,X2):: CALL DISTANCE(#15,#3,
X3)
1540 A=ABS((X1<X2 AND X1<X3)-1):: B=(X3<
X1 AND X3<X2)+3 :: CALL DELSPRITE(#A,#B)
1550 CALL POSITION(#15,Y1,X1):: IF Y1<YS
(E+3)-1 THEN CALL SOUND(10,880,0):: GOTO
 1550 ELSE CALL MOTION(#15,0,0)
1560 IF F+SN=0 THEN 1610
1570 CALL POSITION(#16,Y3,X3):: IF Y3<YS
(E+3)-2 THEN CALL SOUND(10,880,0):: GOTO
 1570
1580 IF F THEN CALL CHAR(37,"00000000011
E0FFF",39,"000000000080402F")ELSE CALL C
HAR(37,"00000000000C3E7F",39,"0000000000
00C0E")
1590 CALL MOTION(#16,0,0)
1600 CALL LOCATE(#15,YS(E+3),X1,#16,YS(E
+3),X3)
1610 FOR I=0 TO 20 :: CALL SOUND(-10,523
-8*I,I,784+8*I,I):: NEXT I
1620 LE=LE-1 :: IF LE<0 THEN 2260 ELSE C
ALL HCHAR(1,19+LE,32)
1630 T=T+4 :: IF F+SN=0 THEN 1300
1640 IF K=0 THEN CALL DELHIND(1,6-A-B,C(
.5+SGN(1-ABS(E-1.5))/2))
1650 CALL PATTERN(#15,122-X/2)
1660 F,SN=0 :: K=SGN(E-.5)
1670 X1=246 :: X2,Y2=1 :: GOSUB 1800 ::
CALL SOUND(30,523,3)
1680 CALL POSITION(#15,Y1,X1):: CALL HCH
AR((Y1+15)/8,32,2):: CALL SPRITE(#14,40,
3,Y1-1,X1-8):: CALL SOUND(20,698,0)
1690 X1=X3 :: GOSUB 1800 :: CALL DELSPRI
TE(#16):: CALL SOUND(20,-5,0):: CALL CHA
R(37,RPT$("0",33))
1700 X1=6 :: GOSUB 1800 :: CALL SPRITE(#
17,116,3,YS(E+3),1):: CALL HCHAR((Y1+7)/
8,1,32,2)
1710 FOR I=-8 TO 8 :: CALL MOTION(#17,I,
0):: CALL SOUND(-10,210-ABS(I),0):: NEXT
 I
1720 CALL DELSPRITE(#17):: CALL HCHAR((Y
1+7)/8,1,116):: CALL HCHAR((Y1+7)/8,2,11
7)
1730 X1=241 :: GOSUB 1800 :: CALL SOUND(
30,698,3)
1740 CALL DELSPRITE(#14):: CALL HCHAR((Y
1+15)/8,32,114):: CALL SOUND(20,523,0)
1750 T=T+3.7 :: IF K1 THEN 1300
1760 X1=XS(2.5+1.5*K):: X2=XS(6-K):: Y2=
YS(6-K):: GOSUB 1800
1770 E=2*K-(B+1)*(K+1)/2 :: IF B+K=-2 TH
EN 2100 ELSE 2210
1800 CALL JOYST(1,X,Y):: CALL PATTERN(#1
5,122-X/2):: CALL MOTION(#14,0,X*2,#15,0
,X*2)
1810 T=T+.34 :: CALL COINC(#15,YS(E+3),X
1,5,A):: CALL COINC(#15,Y2,X2,5-10*(E<1)
,B):: CALL KEY(1,RET,STA):: IF RET+A+B<>
17 THEN 1800
1820 CALL MOTION(#14,0,0,#15,0,0):: RETU
RN
1900 IF K1 OR E=3 THEN 1300
1910 CALL MOTION(#1,0,0,#2,0,0,#3,0,0,#1
5,0,0,#16,0,0):: DISPLAY AT(24,1):"TOILE
TTE"
1920 FOR G=1 TO 15 :: FOR N=1 TO 3 :: CA
LL SOUND(-10,100+10*G*G*N,G):: NEXT N ::
 NEXT G
1930 K1=E :: E=2-SGN(K1-.5):: CALL MOTIO
N(#1,0,V,#2,0,V,#3,0,V):: GOTO 1300
1940 CALL COLOR(#15,1,#16,1):: CALL TUER
(233,81,0):: DISPLAY AT(24,1):""
1950 FOR G=20 TO 0 STEP -1 :: CALL SOUND
(20,523,G):: CALL SOUND(20,659,G):: CALL
 SOUND(20,784,G):: NEXT G
1960 FOR G=0 TO 20 :: CALL SOUND(-10,-5,
G):: NEXT G
1970 CALL TUER(233,81,1):: CALL HCHAR(11
,31,102):: CALL HCHAR(12,31,103):: CALL
```

```
COLOR(#15,5,#16,3+4*SN)
1980 E=ABS(K1)*(K=0)-2*ABS(K):: K1=0 ::
T=T+10
1990 GOTO 1300
2100 CALL MOTION(#1,0,0,#2,0,0,#3,0,0)
2110 CALL POSITION(#15,Y1,X1):: IF X1<24
 THEN 2150 :: X3=INT(X1/8)+2 :: IF S(X3-
5)=0 THEN 2150
2120 CALL CHAR(37,SA$):: CALL HCHAR(12+(
S(X3-5)>2),X3,33.5+1.5*(-1)^S(X3-5)):: S
(X3-5)=S(X3-5)-1
2130 CALL SPRITE(#16,36,7,Y1-16,X1+4,0,0
):: SN=1 :: T=T+3
2140 FOR G=1 TO 2 :: FOR I=0 TO 16 :: CA
LL SOUND(-10,523+I*15,0):: NEXT I :: NEX
T G
2150 E=-2 :: V=-7 :: CALL MOTION(#1,0,V,
#2,0,V,#3,0,V):: GOTO 1300
2200 IF E=-2 OR E=1 THEN CALL MOTION(#1,
0,0,#2,0,0,#3,0,0)
2210 CALL CHAR(128,AM$(.5+ABS(E+.5)),132
,AM$(-1.5+ABS(E+.5)))
2220 IF INT(LOG(T)^4/190)=HW THEN 2280 E
LSE HW=INT(LOG(T)^4/190)
2230 IF HW=INT(HW/8)*8 THEN CALL HCHAR(2
3-HW/8,21,106,12):: CALL CHAR(107,"0"):
 CALL HCHAR(22-HW/8,21,107,12):: GOTO 22
80
2240 CALL CHAR(107,RPT$("00",8-HW+INT(HW
/8)*8)&RPT$("FF",HW-INT(HW/8)*8))
2250 IF SA(1)*4>=HW THEN 2280 ELSE DISPL
AY AT(24,1):"DAS WASSER LJUFT QBER"
2260 FOR N=1 TO 3 :: FOR G=-30 TO 30 ::
CALL SOUND(10,523,ABS(G),659,ABS(G),784,
ABS(G)):: NEXT G :: NEXT N
2270 CALL DELSPRITE(ALL):: GOTO 300
2280 RV=SGN(ABS(ABS(E)-2)-.5)
2290 Y1=YS(E+3)-2*RV :: X1=XS(E+3)
2300 U=Y1-RV*40 :: O=Y1+RV*4
2310 R=-SGN(ABS(E+.5)-2)
2320 E=EX(E+3):: T=T+1.5
2330 IF E=-3 OR(E=1 AND K=0)THEN 2400
2340 IF K=0 THEN CALL DELHIND(3,1,C(ABS(
(E-2)/3))):: T=T+2.8
2350 T=T+5.6 :: X2=C(INT(.5+SGN(E)/2)):
 Y2=YS(E+3):: CALL TUER((X2),(Y2),1):: G
OSUB 2450
2360 RESTORE 2640 :: FOR I=1 TO 3 :: REA
D A,B :: CALL CHAR(A,A$(HIND(T)))
2370 CALL SPRITE(#I,A,B,Y2,X2,0,24*SGN(A
BS(E-.5)-1)):: GOSUB 2450 :: NEXT I
2380 CALL TUER(X2,Y2,0):: GOSUB 2450 ::
IF K+E=0 THEN E=0
2390 K=0 :: IF K1 THEN E=3
2400 V=8*SGN(ABS(E-.5)-1):: CALL MOTION(
#1,0,V,#2,0,V,#3,0,V)
2410 GOSUB 2450 :: GOTO 2410
2450 CALL JOYST(1,X,Y):: Y1=Y1+X*R :: IF
 Y1=0 THEN Y1=Y1-X*R :: GOTO 2480
2460 IF X=0 THEN 2480 ELSE CALL SOUND(10
,500,0)
2470 X1=X1+X :: CALL PATTERN(#15,130-X/2
):: CALL LOCATE(#15,Y1,X1,#16,Y1-16,X1+4
+2*R)
2480 IF Y1=U THEN 2490 :: T=T+.28 :: RET
URN
2490 CALL LOCATE(#15,YS(E+3),X1,#16,YS(E
+3)-16,X1+4)
2500 IF E=-3 THEN 710 ELSE 1300
2600 DATA 33,161,189,81,73,81,1,81,225,1
21,71,121,233,81,32,81,1,2,-2,0,-1,-3
2610 DATA 2,1,95,32,8,1,94,32,13,1,94,24
,13,29,94,4,18,1,94,6,18,11,94,22,23,1,9
4,32,3,1,40,96
2620 DATA 3,3,34,28,4,3,34,28,5,3,34,28,
22,19,35,2,22,10,40,1,22,12,40,1,22,14,4
0,1,22,21,107,12
2630 DATA 11,31,102,1,12,31,103,1,12,32,
114,1,17,32,114,1,11,5,35,2,12,5,33,3
2640 DATA 108,14,136,13,140,7
2700 SUB DELHIND(AS,AS1,XT)
2710 CALL POSITION(#15,Y,M):: CALL TUER(
(XT),Y,1):: CO=0
2720 FOR I=AS1 TO AS+AS1-1 :: CALL POSIT
ION(#I,Y,X)
2730 CALL MOTION(#I,0,20*SGN(SGN((M<XT)+
.5)*(ABS(X-(M+XT)/2)-(M-XT)/2+.5))):: NE
XT I
2740 FOR I=AS1 TO AS+AS1-1 :: CALL COINC
(#I,Y,XT,13,A):: IF A THEN CALL DELSPRIT
E(#I):: CO=CO+1 :: IF CO=AS THEN 2760
2750 NEXT I :: GOTO 2740
2760 CALL TUER((XT),Y,0)
2770 SUBEND
2800 SUB TUER(X,Y,Z)
2810 X=X/8+1 :: Y=Y/8
2820 CALL VCHAR(Y,X,97+3*Z,3):: CALL VCH
AR(Y,X+1,96+3*Z,3):: CALL HCHAR(Y+1,X,96
+5*Z):: CALL HCHAR(Y+1,X+1,98+Z)
2830 SUBEND
```

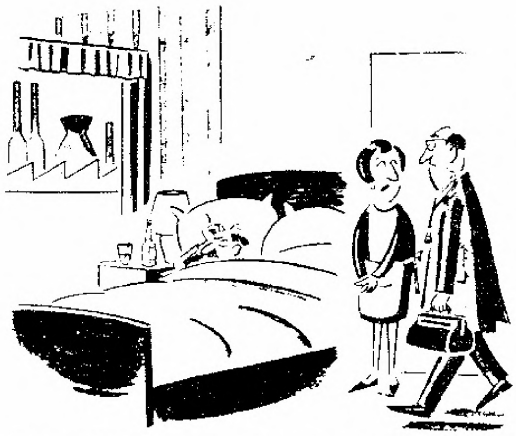
„SCOOTIE": Teil Zwei

```
100 !************************
110 !*                      *
120 !*     SCOOTIE-GRAFIK   *
130 !*                      *
140 !*scootie&scootie-grafik*
150 !*          von         *
160 !*    thorsten spott    *
170 !*  copyright(c) 15.IX.85*
180 !*                      *
190 !************************
200 OPEN #1:"CS1",OUTPUT,SEQUENTIAL,INTE
RNAL,FIXED 65
210 FOR I=1 TO 34 :: READ A$
220 PRINT #1:A$
230 NEXT I
240 CLOSE #1
250 !AM$(0):FIGUR OBERE TREPPE NACH OBEN
260 DATA 000000000000010102040103030700
060B0F0F060F0F8F4F4F4F4F03030307
270 !AM$(1):FIGUR UNTERE TREPPE NACH UNT
EN
280 DATA 060B0F0F060F1F1F2F4F0F0F0D0C0C1
```

. . . nebenbei bemerkt!

Zur Genesung schenkt man oft Blumen . . .

. . . und doch soll es einige Leute geben, die erst mit der COMPUTRONIC wieder so richtig auf die Beine kommen! Dazu meint Hausarzt Dr. Fiebertropf: „Ich verschreibe Ihrem Mann, Frau Bollerhey, noch schnell ein paar Aspirin, verordne ihm 24 Stunden Bettruhe, und kann dem Armen nur wärmstens ein Computronic-Abo empfehlen."



Wußten Sie eigentlich, daß das Abo mit 6 Ausgaben pro Jahr nur 35 Mark – (im Ausland 45 Mark) – kostet?

Nun, Erna Bollerhey weiß es jetzt. Sie hoffentlich auch!!!

Die Abo-Karte finden Sie auf Seite 44!


```
C000000000000808040400 0E0E02
290 !AM$(2):FIGUR OBERE TREPPE NACH UNTE
N
300 DATA 000000000000010102020007070400 0
060D0F0F060F0F8F8F4F2F0F0B0303038
310 !AM$(3):FIGUR UNTERE TREPPE NACH OBE
N
320 DATA 060D0F0F060F1F2F2F2F2F0F0C0C0C0
E000000000000808040208 0C0C0E
330 !AM$(4):FIGUR AUF LEITER
340 DATA 090B0B0B090703030303030303030
380C0C0C088C8E0D0C0E0F03
350 !AM$(5):FIGUR AUF LEITER
360 DATA 010303031113171B03070F0C0000000
090D0D0D090E0C0C0C0C0C0C0C0C0C0C
370 !AM$(6):FIGUR GESTUERZT NACH RECHTS
380 DATA 000000000000000000000030707CFF8B
0000000000000000000EFFFDEEE010080C
390 !AM$(7):FIGUR GESTUERZT NACH LINKS
400 DATA 000000000000000070FFBF770708103
0000000000000000000000000E0E0F31F0D
410 !A$(0):BALL
420 DATA 000000000000000000000000000000
000000000000000000000003C7E7E7E7E3C
430 !A$(1):SKATEBOARD
440 DATA 000000000000000000000000FF10313
1000000000000000000000010E080808
450 !A$(2):HUND
460 DATA 000000000000000008080F0F1F1F342
4000000000000000001836FEFEF0E0505
470 !A$(3):SERVIERWAGEN
480 DATA 000000000000000201F08080F081C0
8000000000000000000FC0808F8081C08
490 !A$(4):KINDERWAGEN
500 DATA 000000003C7C7C7C7F7F7F300D16381
0000000000000608F0F0F06080D0381
510 !CHAR 32-35:LEERZEICHEN,SANDSAECKE,D
ACH,SANDSACK
520 DATA 000000000000000007EFFFF7E7EFFFF7
E007E7E7E7E7E7E00000000007EFFFF7E
530 !CHAR 36-39:GELOESCHT FUER #16
540 DATA 000000000000000000000000000000
000000000000000000
550 !CHAR 40-43:#EIMER
560 DATA 000000000000000000000000000000
000000000000000001C22417F3E3E3E3E
570 !CHAR 44-47:DOPPELPUNKT,MINUS,PUNKT,
BINDESTRICH
580 DATA 001818000018180000000007E7E00000
000000000000018180000626408102646
590 !CHAR 48-51:0,1,2,3
600 DATA 007CFEC6C6C6FE7C001838581818183
C007CFE067CC0FEFE00FEFC081C06FEFC
610 !CHAR 52-55:4,5,6,7
620 DATA 000C1C3C6CFE0C0C00FEFEC0FC06FE7
C007CFEC0FCC6FE7C00FEFC0C1818303
630 !CHAR 56-59:8,9
640 DATA 007CFEC67CC6FE7C007CFEC67E06FE7
C
650 !CHAR 64-67: ,A,B,C
660 DATA 000000000000000000038386C7CEEEEC
600FCFEE6FCE6FEFC003C7EF8E0F07E3C
670 !CHAR 68-71:D,E,F,G
680 DATA 00F8FCFEC6FEFCF800F8F8E0FCE0FEF
E00FEFEE0FCE0E0E0003C7EF8E0FE7E3A
690 !CHAR 72-75:H,I,AE,K
700 DATA 00EEEEFEFEFEEEEE007C7C3838387C7
CC6BA386C7EEEEEC600EEECF8F8F8ECEE
710 !CHAR 76-79:L,OE,N,O
```

```
720 DATA 00E0E0E0E0FEFEFE82BA7CFEEEFE7C3
8008ECEEEFEEEE6E200387CFEEEFE7C38
730 !CHAR 80-83:P,UE,R,S
740 DATA 00FCFEE6FEFCE0E0EE00EEEEEEFEFE7
C00FCFEE6FEFCECEE007CFEF07C1EFE7C
750 !CHAR 84-87:T,U,V,W
760 DATA 00FEFEFE3838383800EEEEEEEEFEFE7
C00EEEE6C7C38381000D6D6D6FEFEEE82
770 !CHAR 88-91:UNTERE TREPPE,OBERE TREP
PE
780 DATA 0F0F0F0FF0F0F0F0F0F0F0F00F0F0F0
F.
790 !CHAR 92-95:LEITER,LEITER,FUSSBODEN,
DACHFIRST
800 DATA FFFFC0C0FFFFC0C0FFFF0303FFFF030
3FFFFFFFF000000000000000FFFFFFFF
810 !CHAR 96-99:TUER GESCHLOSSEN,TUER OF
FEN
820 DATA FFFFFFFFFFFFFFFFFFFFFF7F7F7FFFF
```

TI 99/SCOOTIE

```
FFFFFFFFFFFF1FEFF0
830 !CHAR 100-103:TUERELEMENT, ,TOILETTE
840 DATA 8080808080808080808080808080C0C08
0000000010101413 10D03FFFE7C3C3E3F
850 !CHAR 104-107:KOPF, ,WASSER,WASSER
860 DATA 1834183C7EBEBD00000000000000000
0FFFFFFFFFFFFFFFF0
870 !CHAR 108-111:LOESCHEN FUER #1(HINDE
RNIS)
880 DATA 0000000000000000000000000000000
000000000000000000
890 !CHAR 112-115:MUELLEIMER,EIMER
900 DATA 3F2A2A2515151D1FE0A0A0204040C0C
01C22417F3E3E3E3E0
910 !CHAR 116-119:#MUELLEIMERDECKEL
920 DATA 000000000002053F000000000000000
000000000000000E00
```

ENDE DES LISTINGS

Tips u. Tricks – Tips u. Tricks – Tips u. Tricks – Tips u. Tricks – Tips u. Tricks

Optimierung von Joystickabfrage in BASIC-Programmen

Jetzt auch für den VC-20 und den C-16/116!

Grundsätzlich sind die Joystickabfragen bei den gängigen C (C-64/VC-20/C-16/C-128) untereinander nicht kompatibel. Das Anpassen und Umsetzen in andere BASIC-Dialekte ist unter diesen Umständen recht mühsam und man wird wohl in der Regel völlig neue Programme schreiben müssen.

Der C-16 besitzt wie auch der C-64 zwei Joystickports.

Während man beim C-64 über Umweg-Routinen (Speicherplätze : PEEK (56320) JOY 1 & (56321) JOY 2 zur eigentlichen Abfrage kommt, wartet der C-16 mit nur einem Befehl (PRINT JOY (1),(2) auf.

Zusätzlich kann man mit dem eingebauten Assembler -TED-unter Benutzung der Werte $ FF08 / PEEK (65288) kleine MA.-Progr. einbinden. (Die übrige „Tastatur" bleibt davon unberührt.)

Eine schnelle Kombination (BASIC+MA) zeigt das vorliegende Programm. (TEST-PROGRAMM)

Ein wenig aus dem üblichen Rahmen fällt der VC-20 mit seiner JOYSTICKABFRAGE. Hier werden unterschiedliche System-Routinen (Kontroll-Ports) zur Positionsbestimmung (Richtung & Feuerknopf) herangezogen, die dann mit geeigneter „Software-Unterstützung" abgerufen werden können.

Um diese Programme weitgehend (Überschreiben durch BASIC!) zu schützen, legt man sie zweckmäßig in „freie" Speicherbereiche. Dazu bieten sich z. B. die Plätze im Kassetten-Puffer (VC-20 = $ 033C - 03FB / # 828 - 1019 oder C 16 = $ 0332 - 03F2 / # 818 - 1010) an. Ebenfalls für sogn. USER-SOFTWARE finden wir für den C-16 noch folgende Plätze: $ 065E - 06EB / 1630 - 1771.

In der ZERO-PAGE sind (VC 20 = $ 00FA - FE / # 251 - 254 & C 16 = $ 00D8 - 00E8 / # 216 - 232) noch einige BYTES für den Anwender frei.

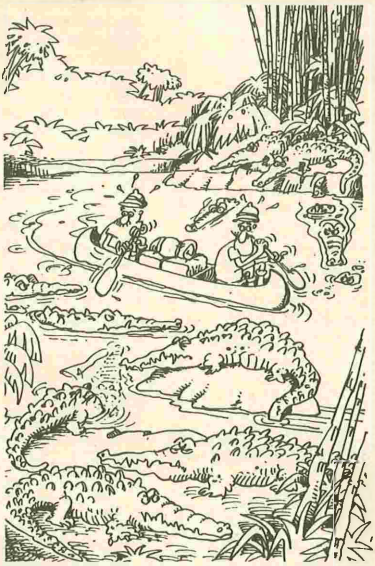
```
1    REM   JOY-TEST  C-16/116
2    GOSUB  100      : REM   BASIC-LADER
3    SCNCLR
4    LET  B = 820
5    POKE  B,JOY  (1)
6    SYS  1630
7    ON PEEK ( 216)  GOSUB  10, 20, 30, 40, 50,
                            60, 70,80, 90
8    POKE 216, Ø
9    GOTO  5
10   PRINT " 1   NORD  "  : RETURN
20   PRINT " 2   NO    "  : RETURN
30   PRINT " 3   OST   "  : RETURN
40   PRINT " 4   SO    "  : RETURN
```

```
<FG> 30170 DATA 83,63,127,255,255,255,25
5,127,63
<HL> 30180 DATA 85,0,1,3,7,15,31,63,127
<NB> 30190 DATA 86,129,0,0,0,0,0,0,0
<DE> 30200 DATA 87,128,0,0,0,0,0,0,128
```

```
<FE> 30210 DATA 28,0,56,116,252,12,140,2
52,120,-1
<FK> 32766 REM ENDLICH !!!......
<KI> 32767 REM PROGRAMMENDE ....?! (&#!@
')
```

Achtung! Achtung! TI-99 Fans – Pitfall-Harry ist wieder unterwegs!!

Freunde von Spielautomaten werden ihn kennen. Nun gibt es Pitfall-Harry auch im TI-Format. Natürlich ist hier Harry wieder auf der Jagd nach den Schätzen des Urwaldes.



Bei diesem Spiel handelt es sich um die TI-Version des alten Spielhallenhits Pitfall. Nach dem Titelbild und der Titelmelodie wird das Spiel mit "Q" oder dem Joystick (1) gestartet. Ihr Weg beginnt mit Bild 1. Die Spielfigur muß jedes Bild von links nach rechts durchqueren, wobei überall Gefahren lauern. Es gibt giftige Schlangen, die ebenso übersprungen werden müssen wie rollende Baumstämme. Im Wasser liegende Krokodile können nur überquert werden, wenn ihre Mäuler geschlossen sind. Sie können natürlich auch über die Krokodile hinwegspringen. In den unterirdischen Gängen lauern Skorpione, die sofort angreifen, wenn man das Stockwerk betritt. Achtung! Je weiter man in den Urwald vordringt, desto schneller werden die Skorpione, so daß man nur durch geschickte Sprünge den Angreifern ausweichen kann. Geldbeutel, die entweder unter der Erde oder am Wege liegen, bringen zusätzliche Punkte. Die Anzahl der überwundenen Bilder wird bei jedem Bildwechsel dem Punktestand des Spielers gutgeschrieben. Hat man alle 5 Leben verloren, wird das Endbild angezeigt. Wenn man alle Bilder bestanden hat, gelangt man in einen Tempel, der einen Schatz in Form eines riesigen Diamanten beherbergt. Nachdem man diesen gehoben hat, wird man von einem Jeep abgeholt und man kann von neuem spielen. (Uwe Knierim)

Steuerung:

- Gehen: Joystick nach links oder rechts
- Klettern: Joystick oben und unten
- Springen: Joystick nach rechts oder links und Feuerknopf betätigen

~~~~~~~~~~~~ Pitfall-Harry * Das Listing ~~~~~~~~~~~~

```
100 ! ************
110 ! *  PITFALL *
120 ! * (c) 1985 *
130 ! *    by    *
140 ! * J.KRAUSZ *
150 ! *  grafik  *
160 ! * A.KELLAS *
170 ! * ti-99/4a *
180 ! * +X-BASIC *
190 ! * +JOYSTIC *
200 ! *12.644  KB*
210 ! ************
220 ! ACHTUNG!
230 ! DIE BILDERANZAHL
240 ! KANN IN ZEILE 710
250 ! VON  >24 AUF EINE
260 ! ANDERE ZAHL
270 ! GEAENDERT WERDEN!
280 CALL CHARSET :: CALL DELSPRITE(ALL):
: CALL CLEAR :: CALL TITLE
290 CALL CLEAR :: RANDOMIZE :: DIM PAT(2
,2):: PAT(0,1)=104 :: PAT(2,1)=96 :: LIF
E=5 :: SCR=1 :: POINTS=0 :: TIME=0
300 CALL MAGNIFY(3)
310 CALL SCREEN(2):: CALL GRAFIK :: CALL
 BILD :: CALL SPRITE(#1,96,5,98,255):: C
ALL SET
320 CALL JOYST(1,JY1,JY2):: CALL POSITIO
N(#1,POS1,POS2):: IF (JY1=-4 OR JY1=4)AN
D JY2=0 THEN 330 ELSE CALL MOTION(#1,0,0
):: GOTO 350
330 CALL PATTERN(#1,PAT(JY1/4+1,JY2/4+1)
+4):: CALL MOTION(#1,0,JY1*2):: CALL SOU
ND(-10,-5,5):: CALL SOUND(-50,-3,0)
340 CALL PATTERN(#1,PAT(JY1/4+1,JY2/4+1)
)
350 CALL KEY(1,K,S):: IF K=18 THEN GOSUB
 500
360 IF POS2>233 THEN CALL COLOR(#1,1):: 
CALL MOTION(#1,0,0):: GOSUB 660 :: CALL
SET
370 IF POS2<18 THEN CALL LOCATE(#1,POS1,
19):: CALL SOUND(-100,110,0,-5,8)
380 IF POS1>120 THEN GOSUB 410 ELSE GOSU
B 460
390 IF JY1=0 AND(JY2=4 OR JY2=-4)THEN ON
 JY2/4+2 GOSUB 540,650,590 :: CALL POSIT
ION(#1,POS1,POS2)
400 GOTO 320
410 ! untere etage
```

```
420 CALL POSITION(#1,PP1,PP2,#2,P21,P22)
:: CALL MOTION(#2,0,SGN(PP2-P22)*7*(SCR+
15)/15):: CALL PATTERN(#2,134+SGN(PP2-P2
2)*2)
430 CALL COINC(#2,#3,15,CC1):: IF CC1 TH
EN RETURN ELSE CALL COINC(ALL,C):: IF NO
T C THEN RETURN
440 CALL COINC(#1,#3,10,C1):: IF C1 THEN
 POINTS=POINTS+100 :: CALL POINTS(POINTS
):: CALL MELODY4 :: RETURN ELSE CALL DEA
D :: GOSUB 690 :: CALL SET :: LIFE=LIFE-
1 :: IF LIFE=-1 THEN CALL END(POINTS,SCR
)
450 GC=32 :: RETURN
460 TIM=TIM+1 :: IF TIM>3 THEN CALL PATT
ERN(#6,116,#7,112,#8,116):: IF TIM=8 THE
N CALL PATTERN(#6,112,#7,116,#8,112):: T
IM=0
470 CALL COINC(ALL,C):: CALL COINC(#2,#3
,13,CC1):: IF C AND CC1 THEN RETURN ELSE
 IF NOT C THEN RETURN
480 CALL COINC(#1,#3,10,C1):: IF C1 THEN
 POINTS=POINTS+100 :: CALL POINTS(POINTS
):: CALL MELODY4 :: RETURN ELSE CALL DEA
D :: GOSUB 690 :: CALL SET :: LIFE=LIFE-
1 :: IF LIFE=-1 THEN CALL END(POINTS,SCR
)
490 GC=32 :: RETURN
500 REM jump <^>
510 IF POS2+JY1*8<0 OR POS2+JY1*8>250 TH
EN RETURN
520 FOR T=-50 TO 50 STEP 15 :: CALL MOTI
ON(#1,T,JY1*3):: CALL SOUND(-90,900-ABS(
T*12),5):: NEXT T :: CALL MOTION(#1,0,JY
1)
530 CALL POSITION(#1,SS1,DD1):: CALL LOC
ATE(#1,POS1,DD1):: CALL SOUND(-100,-1,0)
:: RETURN
540 REM Latter down
550 CALL GCHAR(INT(POS1/8)+4,INT(POS2/8)
+2,GC):: IF GC=81 OR GC=82 THEN 560 ELSE
 580
560 CALL MOTION(#1,20,0):: FOR DELAY=1 T
O 8 :: CALL SOUND(-10,800,4):: CALL SOUN
D(-10,400,3):: NEXT DELAY :: CALL MOTION
(#1,0,0)
570 CALL LOCATE(#1,161,POS2):: CALL SOUN
D(-100,-3,0)
580 RETURN
590 REM latter up
600 CALL GCHAR(INT(POS1/8)-4,INT(POS2/8)
+2,GC):: IF GC=81 OR GC=82 THEN 610 ELSE
 640
610 CALL MOTION(#1,-20,0,#2,0,0):: FOR D
ELAY=1 TO 9 :: CALL SOUND(-10,900,4):: C
ALL SOUND(-10,500,3):: NEXT DELAY :: CAL
L MOTION(#1,0,0)
620 CALL LOCATE(#1,98,POS2):: CALL SOUND
(-100,-3,0)
630 REM
640 RETURN
650 RETURN
660 REM screount
670 SCR=SCR+1 :: DISPLAY AT(1,13)SIZE(3)
:SCR;"L";"L";"L" :: CALL HCHAR(1,15,76)
: CALL HCHAR(1,16+LEN(STR$(SCR)),76)
680 POINTS=POINTS+SCR :: CALL POINTS(POI
NTS):: CALL MELODY1 :: IF SCR>24 THEN CA
LL WIN(POINTS,SCR)ELSE RETURN
690 CALL HCHAR(1,26,76,6-LIFE):: RETURN
700 SUB TITLE
710 CALL CLEAR
720 CALL CHAR(40,"F7F7F7007F7F7F00",41,"
010307001F3F7F00",42,"80C0E000787C7E",43
,"F77737000F070301",44,"F7F6F40070604000
")
730 FOR T=1 TO 4 :: CALL COLOR(T,14,1):
 NEXT T
740 FOR T=5 TO 8 :: CALL COLOR(T,13,1):
 NEXT T
750 CALL COLOR(2,9,1):: CALL SCREEN(11)
760 DISPLAY AT(7,1):"      )*
                +,                    ((
* (( (((( ((,  )(* ((  ((  ((, ((  ((  ((
, ((( ((  (("
770 DISPLAY AT(11,1):" ((  ((  ((  ((  (
(( ((( ((("
780 DISPLAY AT(16,1):" ENTERTAINMENT FOR
 THE TI 99                            W
RITTEN  1985 BY J.KRAUSZ
                 *
790 DISPLAY AT(20,1):" GRAPHICS 1985 BY
A.KELLAS "
800 CALL MELODY2 :: DISPLAY AT(23,1):"
  PRESS FIRE TO BEGIN"
810 CALL KEY(1,K,S):: IF K<>18 AND S<>0
THEN 800 ELSE IF K<>18 THEN 810
820 CALL MELODY3 :: CALL CLEAR :: SUBEND
830 SUB MELODY1 :: CALL MOTION(#2,0,0)
840 FOR T=1 TO 2 :: CALL SOUND(75,784,0)
:: CALL SOUND(75,740,0):: CALL SOUND(-75
,784,0)
850 CALL SOUND(75,880,0):: CALL SOUND(30
0,784,0):: NEXT T
860 CALL SOUND(75,784,0,988,0):: CALL SO
UND(150,784,0,988,0):: CALL SOUND(75,784
,0,988,0):: CALL SOUND(75,880,0,1047,0)
870 CALL SOUND(150,932,0,1109,0):: CALL
SOUND(300,988,0,1175,0):: CALL SOUND(75,
-7,0):: SUBEND
880 SUB GRAFIK
890 CALL CHAR(96,"03070F0707030306060606
0603010303C0E0F040E0C080C0E0E06040C080E0
E0")
900 CALL CHAR(104,"03070F020703010307070
60203010707C0E0F0E0E0C0C06060606060C080C
0C0")
910 CALL CHAR(100,"03070F070703030F3B370
707071F190CC0E0F040E0C08080F0B0808080A0E
0C0")
920 CALL CHAR(108,"03070F0207030101 0F0D0
10101050703C0E0F0E0E0C0C0F0DCECE0E0E0F89
830")
930 CALL CHAR(112,"0000000018287C5E1E070
E014054FF7F0000000000000000CA6D2C2E77FF
FFF")
940 CALL CHAR(116,"0000000000000000000007
05FBFAB6D7F0000000000000000384C86A6CF3
FFF")
950 CALL CHAR(120,"0000000010306 0D1B3F4
693AB9244381C366FFFEF5EBCF8B060C08000000
000")
```

```
960 CALL CHAR(124,"00000F0F0701061C3A7A7
CFEE6F8FE7F00F8FCFCF8E0B01CAEBF1FAFAF1FB
FFC")
970 CALL CHAR(128,"000C377F07070E1810303
F003F003F000000000000000018303020404040C
000")
980 CALL CHAR(132,"01030704E0306F077D1D7
C380000000080E0703070E0E0E05048000000000
000")
990 CALL CHAR(136,"01070E0C0E0707070A140
000000000080C0E020070CF6E0BEB83E1C00000
000")
1000 CALL CHAR(40,"F7F7F7007F7F7F00",41,
"00FFFFFFFFFFFFFF",42,"",64,"87FF9FFF87F
F8FFF",65,"",72,"071F3F7F7FFFFFFF",73,"E
0F8FCFEFEFFFFFF",74,"")
1010 CALL CHAR(67,"00000101010307 1F",68,
"0000808080C0E0F8",69,"00080C0603030100"
,70,"0808183CBDFFFFFF",71,"003060E0C0808
000")
1020 CALL CHAR(91,"071F3F7F7F3F1F07",92,
"E0F8FCFEFEFCF8E0",93,"FFFFFFFFFFFFFFFF"
,66,"1C3E3E7F2A3E361C")
1030 CALL CHAR(74,"FFFFFF7F7F3F1F07",75,
"FFFFFFFEFEFCF8E0",76,"FFFFFFFFFFFFFFFF"
,77,"",80,"",81,"607F7F60607F7F60",82,"0
6FEFE0606FEFE06")
1040 CALL COLOR(2,9,12,6,13,4,5,2,13,7,1
4,1,8,6,12,3,16,13,4,16,13)
1050 SUBEND
1060 SUB BILD
1070 DISPLAY AT(1,1):"L000LLLLLLLLL1LLLL
LLLLLBBBBLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
LLLLLLLLLLLLLLLLLLLLLLLEFGLLLLLLLEFGLLKJL
LL"
1080 DISPLAY AT(6,1):"LLLKMMMJL@KMMMMMJK
J@LKMMMMJLKJKMMMMMJ@MMMMMMMMM@MMMMMMMMM
MMMMMMM@MMMMMMMMMM@MMMMMMMMMMMMMMMHI@MMMM
MMMMM@MMMMMMMM"
1090 DISPLAY AT(10,1):"MMMMMMHLL@IMMMMMM
MM@MMMMMMMMMMMHLLLL@LIHIMMMMH@LIMMMMMHM
MMHLLLLC DLLLIMMHC DLIMMMHL*************
***************"
1100 DISPLAY AT(14,1):"*****************
*************************************)
))))))))))))))))))))))))))))": : :
1110 DISPLAY AT(23,1):"))))))))))))))))
))))))))))" :: CALL HCHAR(20,1,32,96)
1120 SUBEND
1130 SUB SET :: CALL DELSPRITE(#2,#3,#4,
#5,#6,#7,#8,#9,#10,#11,#12)
1140 CALL VCHAR(1,1,76,7):: CALL VCHAR(1
,2,76,7):: CALL VCHAR(1,31,76,6):: CALL
VCHAR(1,32,76,6):: CALL VCHAR(8,1,77,5)
1150 CALL VCHAR(8,2,77,5):: CALL VCHAR(7
,31,77,6):: CALL VCHAR(7,32,77,6):: CALL
 VCHAR(11,31,76,2):: CALL VCHAR(11,32,76
,2)
1160 CALL HCHAR(23,1,41,32)
1170 CALL HCHAR(13,1,42,32):: CALL HCHAR
(14,1,42,32):: CALL HCHAR(15,1,42,32)::
CALL HCHAR(16,1,41,32):: CALL HCHAR(17,1
,32,192)
1180 RANDOMIZE :: IF1=INT(2*RND)+1 :: IF
2=INT(2*RND)+1 :: IF3=INT(2*RND)+1 :: IF
4=INT(1*RND)+1
1190 WO=INT(12*RND)+3 :: WO1=(INT(12*RND
)+1)+13
1200 IF IF1=1 THEN CALL VCHAR(16,WO,81,7
):: CALL VCHAR(16,WO+1,82,7):: CALL HCHA
R(15,WO,32,2)
1210 IF IF2=1 THEN CALL VCHAR(16,WO1,81,
7):: CALL VCHAR(16,WO1+1,82,7):: CALL HC
HAR(15,WO1,32,2)
1220 ON INT(5*RND)+1 GOTO 1230,1230,1230
,1260,1270
1230 CALL DELSPRITE(#2,#3,#4,#5,#6,#7,#8
,#9,#10,#11,#12):: IF IF3=1 THEN CALL SP
RITE(#4,128,2,98,INT(205*RND)+27,0,0)
1240 IF IF4=1 THEN IF INT(2*RND)+1=1 THE
N CALL SPRITE(#3,124,16,98,INT(204*RND)+
17,0,0)ELSE CALL SPRITE(#3,124,11,160,IN
T(204*RND)+17,0,0)
1250 GOTO 1280
1260 FOR T=10 TO 12 :: CALL SPRITE(#T,12
0,7,100,(INT(200*RND)+1)+54,0,-9):: NEXT
 T :: CALL COINC(ALL,C):: IF C THEN 1260
 ELSE 1280
1270 DISPLAY AT(15,9)SIZE(12):"]]]]]]]]]
]]\" :: CALL SPRITE(#6,112,2,105,97,0,0,
#7,112,2,105,121,0,0,#8,112,2,105,145,0,
0):: GOTO 1280
1280 CALL POSITION(#1,P1,P2)
1290 CALL SPRITE(#1,96,5,P1,19,0,0,#2,13
2,16,167,128,0,0):: CALL COINC(ALL,C)::
IF C THEN 1230 ELSE 1300
1300 SUBEND
1310 SUB END(POINTS,BILD)
1320 CALL CLEAR :: CALL DELSPRITE(ALL)
1330 CALL CHARSET :: FOR T=1 TO 14 :: CA
LL COLOR(T,16,5):: NEXT T
1340 DISPLAY AT(8,1):"     G A M E   O V
 E R    "
1350 DISPLAY AT(12,1):"YOUR SCORE :";POI
NTS: :"YOUR LAST SCREEN WAS ":"SCREEN";B
ILD
1360 DISPLAY AT(16,1):"DO YOU WANT TO PL
AY AGAIN?  THEN PRESS THE FIRE-BUTTON"
1370 CALL KEY(1,K,S):: IF S=0 THEN 1370
1380 CALL COLOR(1,16,2):: CALL MELODY2 :
: RUN 290
1390 SUBEND
1400 SUB POINTS(P):: CALL DELSPRITE(#3):
: CALL MOTION(#2,0,0,#1,0,0)
1410 DISPLAY AT(1,1)SIZE(6):P;"L";"L";"L
" :: CALL HCHAR(1,3,76):: CALL HCHAR(1,4
+LEN(STR$(P)),76):: SUBEND
1420 SUB MELODY4 :: FOR T=0 TO 30 STEP 2
.25 :: CALL SOUND(-100,2000,T,3000,T,440
,29):: NEXT T :: SUBEND
1430 SUB DEAD
1440 CALL MOTION(#1,0,0,#2,0,0,#3,0,0,#4
,0,0,#5,0,0,#6,0,0,#10,0,0,#11,0,0,#12,0
,0)
1450 FOR T=3 TO 1 STEP -1 :: FOR Y=440*T
 TO 110*T STEP -80 :: CALL COLOR(#1,INT(
Y/120)):: CALL SOUND(-100,Y,0):: NEXT Y
:: NEXT T :: CALL COLOR(#1,1)
1460 CALL MELODY3 :: SUBEND
1470 SUB MELODY2
1480 DATA 1047,988,784,740,1047,988,659,
587,1047,988,659,587,0,0
```

```
1490 RESTORE 1480
1500 FOR T=1 TO 6 :: READ A,B :: CALL SO
UND(100,A,0,523,3,A*3,7):: CALL SOUND(10
0,B,0,523,3,B*3,7):: CALL SOUND(200,A,0,
523,4,A*3,7):: NEXT T
1510 CALL SOUND(200,740,0,294,4):: CALL
SOUND(100,784,0,392,4):: CALL SOUND(100,
740,0,784,4):: CALL SOUND(100,784,0,392,
0)
1520 CALL SOUND(100,880,0,784,4):: CALL
SOUND(200,784,0,392,4):: SUBEND
1530 SUB MELODY3
1540 FOR T=1 TO 5 STEP .25 :: CALL SOUND
(-100,131*T,0):: CALL SOUND(-100,165*T,0
):: CALL SOUND(-100,196*T,0):: NEXT T ::
 SUBEND
1550 SUB WIN(P,SCR):: CALL DELSPRITE(ALL
)
1560 CALL CLEAR :: CALL SCREEN(2):: DIM
R(2):: R(0)=104 :: R(2)=96
1570 CALL CHAR(80,"0101010303030707",81,
"070F0F0F1F1F1F3F",82,"808080C0C0C0E0E0"
,83,"E0F0F0F0F8F8F8FC",84,"1A16110F384A4
83F")
1580 CALL CHAR(85,"1712233F2F7F6752",86,
"ACB244F88EA989FE",87,"F424E2FEFA3FF325"
)
1590 CALL CHAR(88,"FFFFFFFFFFFFFFFF",89,
"00FFFFFFFFFFFFFF",90,"FF00FF00FF00FF00"
)
1600 CALL CHAR(64,"0000000804221009030F1
F201F1F0F07000000081022844860787C827C7C7
8F0")! DIAMANT
1610 CALL COLOR(5,8,1,7,11,4,8,11,7)
1620 DISPLAY AT(1,1):"LLLLLLLLLLLLLLLLLL
LLLLLLLLLLLLLLLLLLLLLLLKJLLLLLLLLLLLLLLL
LLLLLLLKMMMMMMMMMJLLLLLLLLLLLLLLLKMMMMPX
XRMMMMJLLLLLL"
1630 DISPLAY AT(4,1):"LLLLLLKMMMMMQXXSMM
MMMMJLLLLLLLLLKMMMMMMMXQBXMMMMMMMJKJLLKJ
KMMMMMMMMMXACXMMMMMMMMMMJLMMMMMMMMMPYYYZ
ZYYYRMMMMMMMMJ"
1640 DISPLAY AT(8,1):"MMMMMMMMMQXXXZZXXX
SMMMMMMMMMMMMMMMMPYYYYZZYYYYRMMMMMMMMMM
MMMMMMQXXXXZZXXXXSMMMMMMMMMMMMMMMPYYYYYZ
ZYYYYYRMMMMMMM"
1650 DISPLAY AT(12,1):"MMMMMMMQXXXXXZZXX
XXXSMMMMMMMMMMMMMMPYYYYYYZZYYYYYYRMMMMMM
MMMMMQXXXXXXZZXXXXXXXSMMMMMMMMMMMMMPYYYYYYY
ZZYYYYYYYRMMMMMM"
1660 DISPLAY AT(16,1):"MMMMMQXXXXXXXZZXX
XXXXSMMMMMMTVMPYYYYYYYYZZYYYYYYYYRMTVMM
UWMQXXXXXXXXZZXXXXXXXSMUWM"
1670 CALL HCHAR(19,1,41,32)
1680 CALL VCHAR(1,1,76,6):: CALL VCHAR(1
,2,76,6):: CALL VCHAR(1,31,76,7):: CALL
VCHAR(1,32,76,7)
1690 CALL VCHAR(7,1,77,12):: CALL VCHAR(
7,2,77,12):: CALL VCHAR(8,31,77,11):: CA
LL VCHAR(8,32,77,11)
1700 CALL SPRITE(#1,96,5,129,20)
1710 CALL COLOR(5,14,1,5,8,1):: CALL JOY
ST(1,JY1,JY2):: IF JY1+JY2=0 OR ABS(JY2)
=4 THEN CALL MOTION(#1,0,0):: GOTO 1730
ELSE CALL PATTERN(#1,R(JY1/4+1)+4)
1720 CALL SOUND(-100,-5,5):: CALL PATTER
N(#1,R(JY1/4+1)):: CALL SOUND(-100,-3,0)
:: CALL MOTION(#1,0,JY1*3)
1730 CALL KEY(1,K,S):: IF S<>0 THEN CALL
 JEEP :: CALL END(P,SCR)
1740 IF JY2=4 THEN CALL POSITION(#1,P1,P
2):: CALL GCHAR(10,INT(P2/8)+1,GC):: IF
GC=90 THEN GOSUB 1760
1750 GOTO 1710
1760 CALL MOTION(#1,-20,0):: FOR T=1 TO
15 :: CALL SOUND(1,-1,0):: CALL SOUND(1,
-2,0):: NEXT T :: CALL MOTION(#1,0,0)
1770 CALL LOCATE(#1,34,121)
1780 CALL MELODY3 :: CALL COLOR(5,1,1)
1790 CALL MOTION(#1,20,0):: FOR T=1 TO 1
5 :: CALL SOUND(1,-1,0):: CALL SOUND(1,-
2,0):: NEXT T :: CALL MOTION(#1,0,0)
1800 CALL LOCATE(#1,129,121):: CALL JEEP
 :: CALL END(P+500,SCR)
1810 SUBEND
1820 SUB JEEP
1830 CALL CHAR(120,"010102020505FE87F9FE
E3BDD622361C7C3C5C381CFC7E7FFFFF63DB3622
361C")
1840 CALL SPRITE(#2,120,2,129,250,0,-8)
1850 CALL COINC(ALL,C):: IF C THEN CALL
COLOR(#1,1):: CALL MELODY2 :: CALL CLEAR
 :: GOTO 1860 ELSE CALL SOUND(-100,110,0
,-7,8):: GOTO 1850
1860 SUBEND
```


*** Aufruf an alle Softwareautoren * Aufruf an alle Softwareautoren ***

AUFRUF

Der Tronic Verlag fordert hiermit alle Softwareautoren zur Einsendung ihrer Programmbeiträge auf. Wir suchen interessante Programmvorschläge insbesondere für die Computersysteme C-64/Schneider/VC-20/C-16/Atari/Spectrum/TI-99 und Apple. Wir zahlen als Gegenleistung ein Honorar von 120,– DM pro komplett veröffentlichte Seite.

Programmeinsendungen bitte an:

Tronic Verlag – Postfach 870 – 3440 Eschwege

AUFRUF

*** Aufruf an alle Softwareautoren * Aufruf an alle Softwareautoren ***

Für den T.I-99/4A

Bei diesem Spiel werden Sie »Baukloetze staunen«!

ODYSEE

Mit Extended Basic

Zeigen Sie was man als Ausbrecherkönig des Jahres 3286 alles so drauf hat!

Fliehen Sie aus einem der strengbewachtesten Gefängnisplaneten des Universums und lassen Sie Ihren Häschern, die sich eine Einfangprämie sichern wollen, keine Chance.

Nachdem Sie das Programm gestartet haben, erscheint das Titelbild und Sie können FIRE JOYSTICK 2 drücken. Sie sehen eine Beschreibung der Route. Der leuchtende Punkt ist das Gebiet, indem Sie sich befinden.

Sieben verschiedene Phasen müssen sie überstehen, um an die Freiheit zu gelangen. In der ersten Phase steuern Sie ihre kleine Raumfähre zum oberen Bildrand, wo Ihr Hauptraumschiff wartet. Sie können nur nach rechts oder links lenken und müssen aufpassen, damit sie nicht mit den anderen Raumschiffen kollidieren. In der zweiten Phase müssen Sie ihr Raumschiff über eine Landschaft lenken und die Polizeiraumschiffe abschiessen. Phase drei: Ihre Laserkanonen sind ausgefallen und sie müssen mit ihrem Raumschiff auf den Berg fliegen, dort können Sie es reparieren. Aber aufgepasst, am Berg sind Laserkanonen stationiert, lassen Sie sich nicht treffen. Phase vier ist dasselbe, wie Phase zwei. Schiessen sie die Polizeiraumschiffe ab. In der fünften Phase weichen Sie den Meteoren aus.Phase sechs: Schiessen oder geschossen werden, das ist hier die Frage. Haben sie auch noch Phase sieben (siehe Phase 5) hinter sich gebracht, sind sie frei !!!

Wichtig ! In den Zeilen 1240 und 1260 müssen sie, A B C als (CTRL,), (CTRL A),(CTRL B),(CTRL C) eintippen. Das ganze Programm ist mit NUM einzugeben. Und jetzt an die Arbeit, liebe Freaks, und nicht einschlafen beim Eintippen.

Das Listing * Das Listing * Das Listing

```
100 !ODYSSEE (C)1985 E.S.A.
110 CALL SCREEN(2):: CALL DELSPRITE(ALL)
:: DISPLAY ERASE ALL :: CALL CHAR(96,"00
00F8FCFEFEEE00EE00FEFEFCF800000000FEFEFE
FEE000F800FEFEFEFE")
120 CALL CHAR(100,"0000EEEEEEEEEE00FE003
8383838000000003E7EFEFEE0007800FEFEFCF8"
)
130 CALL CHAR(104,"0000387CFEFEEE00EE00F
EFE7C3800000",112,"88228822882288225 5AA5
5AA55AA55AA77DD77DD77DD77DDFFFFFFFFFFFFF
FFF")
140 CALL CHAR(120,"00000000000000FFFF"):
: FOR @=0 TO 8 :: CALL COLOR(@,6,2):: NE
X1 @ :: CALL HCHAR(9,1,120,32):: CALL HC
HAR(12,1,121,32)
150 DISPLAY AT(4,2):"EXTENDED SOFTWARE A
RTISTS": :TAB(11);"PRESENT" :: CALL HCHA
R(10,1,115,64)
160 DISPLAY AT(10,1):"ssrrqqppjjh'dffbbj
jppqqrrssssrrqqppjjiaeggccjjppqqrrsss"
:: CALL COLOR(9,16,5,10,16,5,11,13,5,12,
8,2)
170 CALL CHAR(64,"3C4299A1A199423C"):: D
ISPLAY AT(17,4):"@ 1985 BY LUC VERHAEGH"
180 CALL KEY(2,[,]):: CALL KEY(3,N,]):
IF ]+\=0 THEN 180 ELSE CALL E("ONE",0)
190 CALL CHAR(44,"88228822882288225 5AA55
AA55AA55AA77DD77DD77DD77DD0",92,"0000000
0000000FFFF"):: CALL COLOR(9,1,1,10,1,1,
11,1,1,12,1,1)
200 CALL CHAR(120,"07050505050407000000
102FC81BEE000000000FE02FA0A7A823E408",12
4,"E0E0E",125,"",126,"",127,""):: CALL A
(23):: CALL D(26,0)
210 CALL CHAR(96,"FFFE3C0C0404040E0F0F1F
1E3C7CFEFFFFF88",99,"01010303070F3FFF",1
00,"FFFFFFFFFFC38181FFFFFFFFFFFFFFF0808
0404040404040626264646C7CFEFFF")
220 CALL CHAR(112,"FFFCF0E0C0C08080FF3F0
F070303010101010101010101070F1F3F3F7F7
F7F",116,"8",117,"00001",118,"0000001818
",119,"00080000400002")
230 CALL CHAR(104,"3C42A1818181423C00000
0003C42A181071820485090808E018040202010
101",108,"C33C",109,"00000000000000FF",1
10,"1",111,"000008")
240 CALL CHAR(128,"000076555576000000005
354212600000003747146700000000707040700
000",132,"")
```

```
250 CALL CHAR(136,"6090F06",137,"",138,"
",139,"",140,"C0D0747D837F837C78F0000000
0000000000F00C83FEF0000")
260 CALL CHAR(36,RPT$("F",64),40,"071820
40FF0F01000",42,"038D7557FFFFC7000")
270 CALL B :: DISPLAY AT(11,22):"h":TAB(
10);"n";TAB(22);"f     v":TAB(14);"w";TAB
(22);;"g":TAB(22);"e":"o";TAB(17);"nv jk
ei":TAB(19);"s ceb  w"
280 DISPLAY AT(17,1):"  fnotv   otuwvon
plqeki n w ugi uw   nio h akrowieeekwo  j
eb       'i f abci beeeb v cddjkiojkvabig
cddebo eepk  n"
290 CALL DELSPRITE(ALL):: CALL HCHAR(21,
1,101,64:: CALL HCHAR(21,10,32,2):: CAL
L HCHAR(22,10,32,2)
300 CALL MAGNIFY(3):: CALL COLOR(9,13,2,
10,6,2,11,13,2):: CALL SPRITE(#2,140,8,1
0,100,0,10)
310 CALL SPRITE(#4,40,11,30,256,0,-RND*3
0,#5,140,6,50,1,0,RND*30,#6,40,10,70,256
,0,-RND*30)
320 CALL SPRITE(#7,140,6,90,1,0,RND*30,#
8,136,7,41,100,0,-RND*30,#9,136,14,61,10
0,0,RND*30)
330 CALL SPRITE(#1,136,8,180,1,0,5)
340 FOR @=0 TO 3 :: CALL COLOR(10,6,1)::
 CALL SOUND(300,131,0,131.1,0,-7,25):: C
ALL COLOR(10,16,1):: CALL SOUND(300,-7,2
5):: NEXT @
350 CALL SOUND(1475,-7,25):: CALL SOUND(
1000,-5,20):: CALL MOTION(#1,-3,0)
360 CALL SOUND(1500,-5,25,110,15,110.1,2
0):: CALL SPRITE(#3,36,2,139,73,1,0):: O
N ERROR 380 :: CALL SOUND(1,-5,25):: CAL
L SOUND(-99,-5,25)
370 CALL HCHAR(20,10,32,2):: CALL DELSPR
ITE(#3):: B=99 :: CALL SOUND(-1,
380 CALL JOYST(2,[,]):: CALL MOTION(#1,-
3,[):: CALL COLOR(#2,8,#1,8):: DISPLAY A
T(1,10):"BONUS";B :: B=B-1 :: CALL COINC
(ALL,[):: IF [ THEN 400
390 CALL COLOR(#2,6,#1,6):: CALL POSITIO
N(#1,\,_):: IF \<5 THEN CALL C("FLEW TOO
 HIGH"):: GOTO 410 ELSE 380
400 CALL COINC(#1,#2,13,@):: IF @ THEN 4
30 ELSE CALL C("COLLISION")
410 CALL MOTION(#1,0,0):: FOR @=\ TO 168
 STEP 10 :: CALL LOCATE(#1,@,_):: CALL S
OUND(-99,-5,25,310-@,15)
420 NEXT @ :: CALL SOUND(-1000,-7,10)::
GOTO 1210
430 DISPLAY AT(1,9):"ALL RIGHT!" :: CALL
 DELSPRITE(#1):: CALL MOTION(#2,0,13)::
FOR @=110 TO 3520 STEP 110 :: CALL SOUND
(-99,-5,25,@,15)
440 @=@+110 :: NEXT @ :: CALL SOUND(-800
,-5,25,110,20):: SC=SC+B
450 CALL CHAR(124,"0000001285205501 0",12
6,"",127,""):: CALL POSITION(#2,[,]):: C
ALL SPRITE(#1,124,3,[,]-4,0,13):: CALL S
OUND(-2850,-5,25,110,20)
460 CALL COLOR(#1,4,#1,9):: CALL POSITIO
N(#2,[,]):: CALL COLOR(#1,4,#1,9):: IF ]
<250 THEN 460 ELSE @$="TWO" :: A=26 ::
C=4
470 CALL E(@$,SC):: CALL COLOR(9,1,1,10,
1,1):: CALL CHAR(124,"F0F")
480 []=10 :: CALL A(22):: CALL D(A,C)::
CALL B :: CALL CHAR(42,"1F271F0483CCD7FF
D7CC83041F271F",112,"0101010101016191937
B3F3F3E4DA48400003C42A181FFE080F0F80C860
20303")
490 CALL CHAR(96,"",97,"0718204850908080
0",99,"E01804020201010 1",116,"0004040607
CF7F1F0F1FFF0F1F244609")
500 CALL CHAR(118,"00000000404048F0F0F8F
0FCF0E0A8",40,"00000000010718FF180701",1
04,"3C42A18181814 23C0F0F1F1E3C7CFEFF0000
00003C42A181FFF88")
510 CALL DELSPRITE(ALL):: CALL MAGNIFY(3
):: DISPLAY AT(19,1):: CALL HCHAR(20,1,1
01,64):: ON ERROR 520 :: CALL SOUND(-1,1
10,20,-5,25):: CALL SOUND(-1,
520 @=-99 :: CALL COLOR(9,13,2,10,16,2)
: CALL SPRITE(#4,104,11,137,64,0,@,#5,96
,16,137,128,0,@,#6,112,9,137,192,0,@)
530 B=60 :: L,S=0 :: CALL SPRITE(#1,140,
8,10,10)
540 CALL JOYST(2,[,]):: CALL MOTION(#1,-
]*3,[):: CALL COINC(#1,#2,14,[:: IF [ T
HEN CALL C("COLLISION"):: GOTO 690 ELSE
CALL POSITION(#1,[,])
550 IF [<10 THEN CALL C("FLEW TOO HIGH")
:: GOTO 690 ELSE IF [>130 THEN CALL C("C
HASHED DOWN"):: GOTO 690 ELSE CALL KEY(2
,@,_)
560 IF _=0 THEN 600 ELSE CALL MOTION(#1,
0,0):: CALL SOUND(-500,500,8,700,8,-5,25
,110,20):: A=([+7)/8 :: C=(]+7)/8+2 :: D
=32-C :: CALL HCHAR(A,C,109,D)
570 CALL HCHAR(A,C,32,D):: GOSUB 680 ::
CALL POSITION(#2,E,F):: IF (E<[-10)+(E>[
+10)+(S=0)THEN 590 ELSE CALL SOUND(-400,
-7,9,110,8,110.1,8)
580 SC=SC+5 :: CALL G(2):: CALL DELSPRIT
E(#2):: S=0
590 ON ERROR 600 :: CALL SOUND(-1,110,20
,-5,25):: CALL SOUND(-1,
600 IF S=0 THEN RANDOMIZE :: CALL PEEK(-
31808,@):: CALL SPRITE(#2,40,@/20+3,@/2+
1,256):: S=1
610 CALL POSITION(#2,E,F):: IF F>]+99 TH
EN CALL MOTION(#2,[],-20):: []=-[] ELSE
CALL MOTION(#2,([-E)/2,-20)
620 DISPLAY AT(1,10):"TIME";B :: B=B-1 :
: L=L-1 :: L=MAX(L,0):: IF B>0 THEN 540
ELSE \=0
630 CALL MOTION(#1,0,0):: GOSUB 670 :: C
ALL DELSPRITE(#2):: CALL C("WELL DONE"):
: CALL F
640 CALL SOUND(-3800,-5,25,110,15):: CAL
L CHAR(124,"000000128520550A")
650 CALL POSITION(#1,E,F):: CALL SPRITE(
#2,124,11,E,F-7):: CALL MOTION(#1,0,20,#
2,0,20)
660 CALL COLOR(#2,7):: CALL POSITION(#1,
E,F):: CALL COLOR(#2,11):: IF F>240 THEN
 IF @$="FOUR" THEN 870 ELSE 700 ELSE 660
670 CALL MOTION(#2,-20,-20):: CALL POSIT
ION(#2,[,]):: IF [>240 THEN RETURN ELSE
670
```

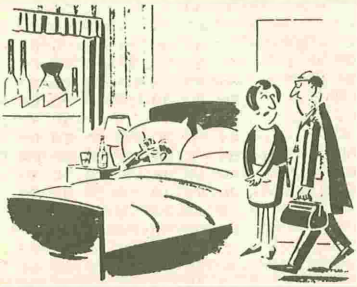
```
680 L=L+2 :: IF L>5 THEN CALL C("LASER O
VERHEATED"):: GOTO 690 ELSE RETURN
690 CALL MOTION(#1,0,0,#2,0,0):: CALL SO
UND(-999,-7,10,110,10,110,1,10):: CALL G
(1):: GOTO 1210
700 B=99 :: CALL E("THREE",SC):: CALL CH
AR(124,"08183060C"):: CALL COLOR(9,2,2,1
0,2,2,11,2,2)
710 CALL A(22):: CALL D(18,10):: CALL B
:: CALL CHAR(96,"03070F1F7F7FFFFF0000000
0000001030F0F3F3FFFFFFFF000000000001030
7",42,"00001FFF57577F7FFFFFFFFFFFFFFFFF"
)
720 CALL CHAR(100,"1F7F7FFFFFFFFFFF00000
0010103071F1F7F7FFFFFFFFFFF00000001071F1
F7F",112,"FFFFFFFFFFFFFFFF",40,"000030FF
",41,"")
730 _=28 :: FOR @=3 TO 10 STEP 7 :: DISP
LAY AT(@,_-1);"cb":TAB(_-2);"cbp":TAB(_-
3);"cbppp":TAB(_-4);"edppp":TAB(_-5);"gfp
ppp":TAB(_-6);"gpppppp":TAB(_-8);"a'pppp
ppp"
740 _=_-8 :: NEXT @:: CALL DELSPRITE(AL
L):: DISPLAY AT(17,11):"cb":TAB(10);"cbp
":TAB(9);"cbpp":TAB(8);"edppp":TAB(7);"g
fpppp" :: FOR @=17 TO 21 :: CALL HCHAR(@
,15,112,18):: NEXT @
750 FOR @=10 TO 16 :: CALL HCHAR(@,23,11
2,10):: NEXT @ :: CALL VCHAR(3,31,112,10
):: CALL VCHAR(2,32,112,10):: CALL HCHAR
(1,31,97):: CALL HCHAR(1,32,96)
760 CALL CHAR(136,"00000055"):: CALL MAG
NIFY(3:: CALL HCHAR(2,30,97):: CALL HCH
AR(2,31,98):: CALL HCHAR(22,9,112,25):: 
CALL HCHAR(22,8,96)
770 CALL COLOR(9,13,2,11,13,2,10,16,2):: 
 CALL SPRITE(#1,140,8,100,30):: ON ERROR
 780 :: CALL SOUND(-1,110,15,-5,25):: CA
LL SOUND(-1,
780 FOR @=6 TO 9 :: CALL SPRITE(#@,40,1,
177,@*16):: NEXT @ :: FOR @=10 TO 14 :: 
CALL SPRITE(#@,40,13,1,239,14,-16):: GOS
UB 810 :: GOSUB 810 :: GOSUB 810 :: NEXT
 @
790 GOSUB 810 :: IF S THEN 800 ELSE RAND
OMIZE :: _=INT(RND*5)+10 :: CALL POSITIO
N(#_,[,]):: IF [>150 THEN 790 ELSE CALL
SPRITE(#2,136,13,[+9,]-8,0,-40):: S=1 ::
 GOTO 790
800 CALL POSITION(#2,[,]:: IF ]<40 THEN
 S=0 :: CALL DELSPRITE(#2):: GOTO 790 EL
SE 790
810 CALL JOYST(2,[,]:: CALL MOTION(#1,]
*-3,0):: CALL POSITION(#1,[,]:: IF [<16
 THEN CALL C("FLEW TOO HIGH"):: GOTO 690
820 IF [>170 THEN CALL C("FLEW TO LOW"):
: GOTO 690 ELSE CALL COINC(#1,#2,12,[::
 IF [ THEN CALL C("COLLISION"):: GOTO 69
0 ELSE B=B-1 :: IF B<0 THEN 840
830 DISPLAY AT(1,10):"TIME";B :: RETURN
840 IF @$=" SIX" THEN 1130 ELSE CALL DEL
SPRITE(#2,#10,#11,#12,#13,#14,#15)
850 CALL MOTION(#1,0,0):: CALL SOUND(-20
00,523,10,659,10,784,10,-5,20):: CALL C(
"THATS GREAT"):: FOR @=1 TO 70 :: CALL
COLOR(#1,5,#1,8)
860 NEXT @ :: CALL DELSPRITE(#1):: DISPL
AY ERASE ALL :: SC=SC+75 :: CALL CHAR(10
1,"FFFFFFFFFFFFFFFF"):: A=18 :: C=20 ::
@$="FOUR" :: GOTO 470
870 CALL E(" FIVE",SC):: CALL COLOR(9,1,
1,10,1,]:: CALL CHAR(124,"80808080808")
:: B=99 :: CALL A(23):: CALL D(8,26)
880 CALL B :: CALL CHAR(136,"02373F7F7F7
F7A58",140,"080C14142626262E1E0E35157535
F5CA0",143,"00000080C0C0C0C")
890 FOR @=0 TO 200 :: NEXT @ :: CALL DEL
SPRITE(ALL):: CALL HCHAR(19,1,32,64):: C
ALL MAGNIFY(3):: CALL COLOR(10,16,2):: C
ALL SPRITE(#1,140,8,150,120)
900 FOR @=3 TO 18 :: CALL SPRITE(#@,136,
INT(RND*14)+3,193,(@-2)*16,10+RND*20,0):
: GOSUB 920 :: NEXT @
910 GOSUB 920 :: GOTO 910
920 CALL JOYST(2,[,]:: CALL MOTION(#1,0
,[*3):: CALL SOUND(-999,-5,25,110,15):: 
CALL COINC(ALL,[):: IF [ THEN CALL C("CO
LLISION"):: GOTO 690
930 B=B-1 :: DISPLAY AT(1,10):"TIME";B :
: IF B>0 THEN RETURN
940 CALL MOTION(#1,0,0):: CALL CHAR(136,
"**"):: CALL C("FANTASTIC!"):: CALL F
950 CALL CHAR(124,"0A110A0012040208"):: 
CALL POSITION(#1,[,]:: CALL MOTION(#1,-
30,0)
960 SC=SC+75 :: CALL SPRITE(#2,124,4,[+1
0,],-30,0):: CALL SOUND(-2000,-5,25,110,
20)
970 CALL COLOR(#2,4,#2,9):: CALL POSITIO
N(#1,[,]:: CALL COLOR(#2,4,#2,9):: IF [
>30 THEN 970 ELSE IF Z THEN 1180 ELSE B=
50 :: @$=" SIX" :: CALL CHAR(124,"01",12
6,"FC")
980 K,S,E=0 :: CALL E(@$,SC):: CALL COLO
R(9,2,2,10,2,2):: CALL A(22):: CALL D(10
,0):: CALL B
990 CALL CHAR(40,"0000E840E36CFCD6EBF57F
EF40E8000000000000E0180402E2FFF",136,"FF
"):: CALL DELSPRITE(ALL):: CALL HCHAR(19
,1,32,64)
1000 CALL CHAR(140,"00000F30C17F0F000000
0000000000030B2EBEC1FEC13E1E0F"):: CAL
L MAGNIFY(3)
1010 CALL COLOR(10,16,1:: CALL SPRITE(#
1,140,8,90,230)
1020 ON ERROR 1030 :: CALL SOUND(-1,110,
15,-5,25):: CALL SOUND(-1,
1030 GOSUB 810 :: IF S=0 THEN CALL SPRIT
E(#3,40,INT(RND*14)+3,1,99,127,0):: GOSU
B 1170 :: CALL MOTION(#3,0,0):: S=1
1040 IF K=0 THEN CALL POSITION(#3,[,]::
 CALL SPRITE(#2,136,6,[+3,116,0,30):: K=
1
1050 CALL KEY(2,[,]:: IF ] AND E=0 THEN
 CALL POSITION(#1,[,]:: CALL SPRITE(#4,
136,12,[+5,]-8,0,-30):: E=1
1060 CALL COINC(#4,#3,18,@):: IF @ THEN
1090 ELSE CALL COINC(#1,#2,10,@):: IF @
THEN 1120 ELSE CALL COINC(#2,#4,18,@)::
IF @ THEN 1110
1070 CALL POSITION(#2,[,]:: IF ]>230 TH
EN CALL SPRITE(#2,32,1,193,1):: K=0 :: G
```

```
OTO 1030 ELSE CALL POSITION(#4,I,J):: IF
 J<99 THEN CALL DELSPRITE(#4):: E=0
1080 GOTO 1030
1090 SC=SC+10 :: CALL MOTION(#1,0,0):: C
ALL SOUND(-999,-7,5,110,10):: CALL COLOR
(#4,1,#2,J):: CALL G(3):: K,S,E=0
1100 CALL SPRITE(#4,32,1,193,128):: GOTO
 1020
1110 SC=SC+5 :: CALL MOTION(#2,0,0):: CA
LL SOUND(-999,-7,5,523,10):: CALL COLOR(
#4,J):: CALL G(2):: K,E=0 :: GOTO 1100
1120 CALL DELSPRITE(#4):: GOTO 690
1130 CALL DELSPRITE(#2,#3,#4):: CALL MOT
ION(#1,0,0):: CALL C("WHAT A HERO!"):: C
ALL F :: CALL CHAR(136,"40AA04A148"):: C
ALL POSITION(#1,I,J)
1140 CALL SOUND(-3000,-5,25,110,10):: CA
LL SPRITE(#2,136,12,I+3,J+18,0,-20)
1150 CALL MOTION(#1,0,-20):: CALL COLOR(
#2,7):: CALL POSITION(#1,I,J):: CALL COL
OR(#2,11):: IF J>30 THEN 1150 ELSE B=99
:: CALL E("SEVEN",SC)
1160 Z=1 :: CALL COLOR(9,1,1,10,1,1):: C
ALL CHAR(126,"",124,"000202020202"):: CA
LL A(23):: CALL D(0,0):: GOTO 880
1170 FOR @=0 TO 30 :: NEXT@ :: RETURN
1180 CALL DELSPRITE(ALL):: CALL COLOR(10
,1,1):: DISPLAY AT(6,1):"  ''''  '''  '''
' '''' ' ''
1190 DISPLAY AT(7,1):"  '     '  ' '     '
   ' '     '''  '''  '''  '''  ' '
'    '  ' '    '            '    '  ' ''
'' '''' ' ''
1200 CALL COLOR(9,6,6,10,16,1):: CALL SP
RITE(#1,140,8,131,120):: @$="YOU MADE IT
!" :: \=9 :: GOTO 1220
1210 @$="GAME OVER!" :: \=10
1220 FOR @=0TO 500 :: NEXT@ :: CALL DE
LSPRITE(ALL):: S$=STR$(SC):: DISPLAY ERA
SE ALL AT(9,\):@$: :TAB(10);"SCORE ";RPT
$("0",4-LEN(S$));S$: :"AGAIN? PRESS FIRE
 ON STICK 2"
1230 CALL COLOR(2,2,2,8,2,2,13,2,2):: FO
R @=6 TO 15 STEP 8 :: CALL HCHAR(@,1,92,
32):: CALL HCHAR(@+1,1,47,32):: CALL HCH
AR(@+2,1,93,32)
1240 DISPLAY AT(@+1,1):"//,,,---------
-,,,//" :: NEXT @ :: CALL COLOR(2,
5,13;8,8,2,13,16,5):: SC,B,L,S,E,K,Z=0 :
: GOTO 180
1250 SUB A(@):: CALL COLOR(2,2,2,8,2,2,1
3,2,2):: CALL HCHAR(@,1,92,32):: CALL HC
HAR(@+1,1,47,32):: IF @<23 THEN CALL HCH
AR(@+2,1,93,32)
1260 DISPLAY AT(@+1,1):"//,,,---------
-,,,//" :: CALL COLOR(2,5,13,8,8,2
,13,16,5):: SUBEND
1270 SUB B :: FOR @=0TO 30 :: RANDOMIZE
 :: CALL PEEK(-31808,I,J):: I=I/12+3 ::
J=J/15+1 :: CALL HCHAR(J,I,110):: NEXT @
 :: SUBEND
1280 SUB C(@$):: DISPLAY AT(1,(28-LEN(@$
))/2):@$;:: SUBEND :: SUB D(I,J):: CALL
MAGNIFY(4):: CALL SPRITE(#1,120,8,80,112
)
1290 CALL SPRITE(#2,124,1,80+I,112+J)::
FOR @=0 TO 3 :: CALL COLOR(#2,J):: GOSUB
 1300 :: CALL COLOR(#2,5):: GOSUB 1300 :
: NEXT @ :: SUBEXIT
1300 CALL SOUND(-1,1000,7,1009,7):: FOR
_=0 TO 30 :: NEXT _ :: RETURN :: SUBEND
:: SUB E(_$,SC):: CALL DELSPRITE(ALL)::
DISPLAY ERASE ALL AT(19,10):"PHASE ";_$:
TAB(11);"SCORE";SC
1310 SUBEND :: SUB F :: FOR @=0TO 7 ::
CALL COLOR(#1,5):: CALL SOUND(-99,523,10
,-5,25):: CALL COLOR(#1,8):: CALL SOUND(
-99,1047,10,-5,25):: NEXT @ :: SUBEND
1320 SUB G(@):: CALL PATTERN(#@,116):: C
ALL COLOR(#@,16,#@,8,#@,12,#@,11,#@,10,#
@,9,#@,7,#@,7,#@,1:: SUBEND
```


...nebenbei bemerkt!

Zur Genesung schenkt man oft Blumen...

... und doch soll es einige Leute geben, die erst mit der COMPUTRONIC wieder so richtig auf die Beine kommen! Dazu meint Hausarzt Dr. Fiebertropf: »Ich verschreibe Ihrem Mann, Frau Bollerhey, noch schnell ein paar Aspirin, verordne ihm 24 Stunden Bettruhe, und kann dem Armen nur wärmstens ein Computronic-Abo empfehlen.«



Wußten Sie eigentlich, daß das Abo mit 6 Ausgaben pro Jahr nur 35 Mark – (im Ausland 45 Mark) – kostet?

Nun, Erna Bollerhey weiß es jetzt. Sie hoffentlich auch!!!

Die Abo-Karte finden Sie auf Seite 44!

```
<AJ> 21770 DATA FF        k  k        FF
<BN> 21780 DATA Ga     n k  k n     aF
<DL> 21790 DATA FFF FFF k  k FFF FFF
<HH> 21800 DATA F     %F        F%     F
<PO> 21810 DATA F     &F aaaa F&     F
<BL> 21820 DATA FFF FFFJJJJJJFFF FFF
<OO> 21830 DATA F%                   %F
<FB> 21840 DATA F&   b I bb I b    &F
<MC> 21850 DATA FFJJJJJJFFFFJJJJJJFF
<HN> 21860 DATA F         %FF%        F
<EC> 21870 DATA Fa        &FF&       aF
<HN> 21880 DATA FFF F FFFFFFFF F FFF
<AO> 21890 DATA F#     %F#  $F%     $F
<HA> 21900 DATA FZ     &FZ  !F&     !F
<ND> 21910 DATA F    FFFF     FFFF    F
<MB> 21920 DATA F    %F#        $F%   F
<PI> 21930 DATA F    &FZ        !F&   F
<PG> 21940 DATA F FFFF  *J1^   FFFF F
<PC> 21950 DATA F %F#   [:=\     $F% F
<DE> 21960 DATA F &FZ   _OIK     !F& F
<HM> 21970 DATA FFFFeeeeeeeeeeeeFFF9
<HO> 21975 DATA 18,22,2,4,1,7,8,2,11,12,
11,0,0,0,8
```

```
<FN> 21980 DATA FFFFFFF%F%FFFFFOFFFF
<PC> 21990 DATA F%FFFF% U %FFF%C%FFF
<ML> 22000 DATA F %FF% #U$ %F% C %FF
<PK> 22010 DATA F  %%  XHY  % .C  %F
<MB> 22020 DATA G           a   C   F
<EA> 22030 DATA FFFFFFFF FFFFFCFFFF
<LK> 22040 DATA F%%%%%%% %FF   C  %F
<MH> 22050 DATA F&b b b  bFF $ C a&F
<KB> 22060 DATA FFFJJJJJJFFFFFFCFFFF
<AO> 22070 DATA F%%       %%F%  n   F
<FP> 22080 DATA Fnn       n&F&   a  F
<GB> 22090 DATA FFFJJJJJJFFFFFF FFFF
<JK> 22100 DATA F%%       %%F       %F
<PP> 22110 DATA F&1       a F ⊞   a&F
<FL> 22120 DATA FFFJJJJJJFFFFFF FFFF
<JN> 22130 DATA F%%       %%F%       F
<JH> 22140 DATA Faa        &F&    $ F
<EG> 22150 DATA FFFFFFFFFFFFFFF FFFF
<MG> 22160 DATA F                    %F
<NI> 22170 DATA F 1   a    b    a &F
<HH> 22180 DATA FFFF FF F FFF FFFFFF
<BH> 22190 DATA FFFeeeeeeeeeeeeeeFI0
<JH> 22200 DATA 1,20,1,6,1,3,10,3,13,3,1
6,15,11,1,6
```

Versetzen Sie sich in diesem tollen Labyrinthspiel in die Rolle des berühmten Forschers Howard Carter, der gerade die Gruft des legendären Tutanchamon entdeckt hat und nun aufbricht, das Grab in allen Einzelheiten zu untersuchen.

TI-99

Tutanchamon's Grab

TI-99

Natürlich ist das nicht so einfach, wie es auf den ersten Blick den Anschein hat. Schließlich muß der Forscher nicht nur die umherliegenden Schlüssel aufsammeln, die Ihm den Zugang zu den Türen eröffnen, auf der Erkundungstour begegnen Ihm auch noch die Skelette vormaliger Grabräuber, die nach einem seiner vier (Spieler-) Leben trachten. Carter hat je doch noch einen Trumpf in der Hand: Wenn die Situation zu brenzlig wird, kann er mit Hilfe der Aid-Taste an den linken oberen Eingang der Gruft fliehen. Dies ist jedoch nur möglich, wenn sich Carter in Bewegung befindet. Hängt er an einer Liane, steht er auf der Leiter oder rutscht er eine der Stangen herunter, wird eine Flucht unmöglich.

Das Spiel bietet vier verschiedene Motive. In einen schwierigeren Level gelangt man erst, nachdem man alle vier Türen geöffnet hat.

Die Steuerung erfolgt über Joystick (Port 2).

Extended Basic ist erforderlich.

```
260 !***********************
270 GOTO 310
280 CALL CHAR :: CALL COINC :: CALL DELS
PRITE :: CALL GCHAR :: CALL HCHAR :: CAL
L JOYST :: CALL KEY :: CALL LOCATE :: CA
LL MOTION :: CALL PATTERN :: CALL SOUND
:: CALL SPRITE :: CALL VCHAR
290 A,A$,HI,HIT,K,KEY,LV,M,P,Q,QQ,S,SCR,
T,TEK,TEK1,X,XJ,Y,YJ
300 DATA
310 RANDOMIZE :: OPTION BASE 1 :: DIM D(
4):: DEF H(X)=(X+7)/8
320 CALL MAGNIFY(3):: CALL CLEAR :: CALL
 SCREEN(2):: CALL COLOR(0,2,7,1,7,2,9,12
,6,10,12,11,11,16,2,12,5,2,13,16,2)
330 !@P-
```

```
100 !***********************
110 !* TUTANCHAMON'S TOMB  *
120 !*    Version 2.3       *
130 !*----------------------*
140 !*  For the TI-99/4A    *
150 !*    Home Computer     *
160 !* with Extended Basic *
170 !*     and Joystick     *
180 !*      in Port 2       *
190 !*----------------------*
200 !*  Size:12659 Bytes    *
210 !*----------------------*
220 !*  By Remco Polman     *
230 !*  Onderste Wehr 43    *
240 !*  6433LC Hoensbroek   *
250 !*      HOLLAND         *
```

```
340 FOR T=3 TO 8 :: CALL COLOR(T,16,2)::
 NEXT T
350 CALL CHAR(40,"C3FFC3C3C3FFC3C33C1008
1008100810",43,"181818181818181C22221C
08180838")
360 CALL CHAR(34,"FFF8E131BBA1D1F1E5E4E5
E7EBDD11",36,"FE282828282828384438282828
282828FFF8E41CCEF3FCFFFFOFB73B78E71FFF")
370 A$="66666666" :: CALL CHAR(58,"7E7E"
&RPT$("18",28),70,A$&A$,74,A$&"667E3C18"
,81,"3C7E"&A$&"7E7E",87,A$&"767676766E6E
6E6E"&A$,100,"",90,A$&"66667E7E")
380 CALL CHAR(112,"1C3E73606060606060606
06063733E1C4242424266667E7E0E0E0408",116
,"3C7E"&A$&A$&A$&"7E3C1C3E736360703C1E07
03030363677E3C")
390 CALL CHAR(96,"0103070F1F3F7FFF80C0E0
F0F8FCFEFF071F3F7F7FFFFFFFFFFFFF7F7F3F1F
07",102,"E0F8FCFEFEFFFFFFFFFFFFFEFEFCF8E
000"&RPT$("F",30))
400 FOR T=0 TO 16 STEP 4 :: CALL CHAR(T+
126,"",T+127,"")):: NEXT T
410 A$="1C7F3A3F0E081C3E" :: CALL CHAR(1
28,A$&"3E3E3E1C0808081E",132,A$&"7EBFBE1
C5474444F")
420 A$="38FE5CFC7010387C" :: CALL CHAR(1
36,A$&"7C7C7C3810101078",140,A$&"7EFD7D3
82A2E22F2")
430 CALL CHAR(60,"00001000080004100008000
1000080018",62,"",63,"",124,"1C7F3E7F5D4
95D3E3E3E3E1C222214FF")
440 CALL CHAR(64,"3C4299919199423C",92,"
0000181818FFFFFF1818181818187EFF",94,"",
95,"")
450 !*************************
460 HI=MAX(P,HI):: CALL DELSPRITE(ALL)::
 P,KEY=0 :: M=4 :: Q,SCR,LV=1 :: GOTO 12
00
470 CALL DELSPRITE(#1):: CALL HCHAR(1,1,
31,768):: CALL VCHAR(1,31,33,96):: FOR T
=1 TO 22 STEP 3 :: CALL VCHAR(T,2,32,2):
: CALL VCHAR(T,31,32,2):: NEXT T
480 FOR T=1 TO 4 :: D(T)=6 :: NEXT T
490 DISPLAY AT(12,9)SIZE(14):"UP TO SCRE
EN";SCR :: DISPLAY AT(14,8)SIZE(16):USIN
G "######## POINTS":P :: DISPLAY AT(16,1
2)SIZE(8):"LEVEL";LV :: RANDOMIZE :: ON
SCR GOSUB 1110,1130,1150,1170500 FOR X=1
 TO 22 STEP 3 :: READ A$ :: DISPLAY AT(X
,1):A$
510 IF 9-(X+2)/3>D(SCR)THEN 560 ELSE A=1
520 Y=POS(SEG$(A$,1,28),"    ",A)
530 IF Y<25 AND SEG$(A$,Y+55+A,1)="!" TH
EN CALL HCHAR(X,Y+2,44)ELSE A=A+1 :: GOT
O 520
540 RANDOMIZE :: Y=INT(RND*24+4)
550 CALL GCHAR(X,Y,TEK):: CALL GCHAR(X-1
,Y,TEK1):: IF TEK<>32 OR TEK1<>33 THEN 5
40 ELSE CALL HCHAR(X,Y,36):: CALL HCHAR(
X+1,Y,37)
560 NEXT X
570 CALL HCHAR(1,30,44):: GOSUB 1820 ::
GOSUB 1830 :: FOR T=1 TO 6
580 Q=-Q :: X=(RND*5+3*LV)*Q :: CALL SPR
ITE(#T*2,120,RND*13+3,T*24+1,100,0,X)::
IF LV>2 THEN CALL SPRITE(#T*2+1,120,RND*
13+3,T*24+1,225,0,X)
590 NEXT T :: IF SCR>1 THEN X=1 :: Y=25
:: CALL SPRITE(#1,124,11,X,Y):: GOTO 910
600 Y=(INT(RND*28+3))*8-7 :: CALL GCHAR(
1,H(Y),TEK):: IF TEK<>32 THEN 600
610 X=1 :: CALL SPRITE(#1,128,11,X,Y)
620 CALL COINC(ALL,HIT):: IF HIT THEN 16
40
630 CALL KEY(2,K,S):: IF S THEN 780
640 CALL JOYST(2,YJ,XJ):: IF XJ+YJ=0 THE
N 620
650 IF XJ THEN CALL PATTERN(#1,124):: GO
TO 920
660 CALL COINC(ALL,HIT):: IF HIT THEN 16
40
670 CALL GCHAR(H(X),H(Y+2*YJ),TEK):: IF
TEK>32 THEN 720
680 Q=-Q :: QQ=8*(YJ=-4):: IF Q=1 THEN C
ALL PATTERN(#1,128-QQ)ELSE CALL PATTERN(
#1,132-QQ)
690 CALL MOTION(#1,0,2*YJ)
700 Y=Y+2*YJ :: CALL MOTION(#1,0,0):: CA
LL LOCATE(#1,X,Y):: IF TEK=43 THEN 880
710 CALL SOUND(1,-5,0):: GOTO 620
720 CALL MOTION(#1,0,0)
730 CALL COINC(ALL,HIT):: IF HIT THEN 16
40
740 IF TEK>39 AND TEK<44 THEN 680
750 IF TEK=44 THEN CALL SOUND(200,880,0,
990,0,1100,0):: KEY=KEY+1 :: P=P+10 :: C
ALL HCHAR(H(X),H(Y+2*YJ),32):: GOSUB 182
0 :: GOTO 680
760 IF TEK=36 AND KEY>0 THEN KEY=KEY-1 :
: D(SCR)=D(SCR)-1 :: CALL SOUND(200,110,
0,220,0,330,0):: P=P+20 :: CALL VCHAR(H(
X),H(Y+2*YJ),32,2):: GOSUB 1820 :: GOTO
680
770 GOTO 620
780 IF K-7 THEN 810
790 CALL PATTERN(#1,60):: FOR T=1 TO 50
:: NEXT T
800 CALL DELSPRITE(#1):: FOR T=0 TO 30 :
: CALL SOUND(-1,-5,30-T):: NEXT T :: Y=4
1 :: CALL SPRITE(#1,60,11,1,Y):: FOR T=1
 TO 50 :: NEXT T :: GOTO 610
810 CALL COINC(ALL,HIT):: IF HIT THEN 16
40 ELSE IF YJ=0 OR(Y<25 AND YJ<0)OR(Y>23
2 AND YJ>0)THEN 870
820 CALL GCHAR(H(X),H(Y+2*YJ),TEK):: IF
TEK=33 OR TEK=36 THEN CALL MOTION(#1,-10
,0):: GOTO 850 ELSE CALL MOTION(#1,-3,2*
YJ):: CALL GCHAR(H(X),H(Y+4*YJ),TEK)
830 CALL COINC(ALL,HIT):: IF HIT THEN 16
40
840 IF TEK=33 OR(TEK=36 AND KEY=0)THEN C
ALL MOTION(#1,3,-YJ/2):: GOTO 660 ELSE 8
60
850 CALL MOTION(#1,0,0):: CALL COINC(ALL
,HIT):: IF HIT THEN 1640 ELSE CALL LOCAT
E(#1,X,Y):: CALL SOUND(-10,-6,0):: GOTO
620
860 Y=Y+2*YJ :: CALL MOTION(#1,1,YJ):: G
OTO 660
870 CALL MOTION(#1,-10,0):: CALL PATTERN
(#1,124):: YJ=0 :: CALL MOTION(#1,0,0)::
 CALL COINC(ALL,HIT):: IF HIT THEN 1640
```

```
ELSE CALL LOCATE(#1,X,Y):: CALL SOUND(-1
0,-6,0):: GOTO 640
880 S=880 :: CALL PATTERN(#1,124)
890 CALL COINC(ALL,HIT):: IF HIT THEN 16
40 ELSE S=S-20 :: CALL SOUND(-1000,S,0,S
+110,0,S+220,0)
900 CALL GCHAR(H(X)+3,H(Y),TEK):: X=X+8
:: CALL LOCATE(#1,X,Y):: IF TEK-33 THEN
890 ELSE CALL SOUND(-10,-6,0):: GOTO 620
910 CALL GCHAR(H(X-XJ/4),H(Y),TEK):: IF
TEK=32 THEN 620
920 CALL COINC(ALL,HIT):: IF HIT THEN 16
40
930 IF XJ=4 THEN 1000
940 IF X=177 THEN 1670
950 CALL GCHAR(H(X+16),H(Y),TEK):: IF TE
K<>42 AND TEK<>40 THEN 1060
960 X=X+8 :: CALL MOTION(#1,5,0):: IF TE
K<41 THEN CALL SOUND(-50,880,0,990,0,110
0,0):: GOTO 990
970 FOR T=30 TO 0 STEP -10 :: CALL COINC
(ALL,HIT):: IF HIT THEN 1640
980 CALL SOUND(-40,-5,T):: NEXT T
990 CALL COINC(ALL,HIT):: IF HIT THEN 16
40 ELSE CALL MOTION(#1,0,0):: CALL LOCAT
E(#1,X,Y):: GOTO 1060
1000 IF X=1 THEN IF SCR>1 AND Y=25 THEN
SCR=SCR-1 :: GOTO 1810 ELSE 620
1010 CALL GCHAR(H(X-8),H(Y),TEK):: IF (T
EK>32 AND TEK<40)OR TEK>42 THEN 1060
1020 X=X-8 :: CALL MOTION(#1,-5,0):: IF
TEK<41 THEN CALL SOUND(-50,880,0,990,0,1
100,0):: GOTO 1050
1030 FOR T=30 TO 0 STEP -10 :: CALL COIN
C(ALL,HIT):: IF HIT THEN 1640
1040 CALL SOUND(-40,-5,T):: NEXT T
1050 CALL COINC(ALL,HIT):: IF HIT THEN 1
640 ELSE CALL MOTION(#1,0,0):: CALL LOCA
TE(#1,X,Y)
1060 CALL COINC(ALL,HIT):: IF HIT THEN 1
640 ELSE CALL JOYST(2,YJ,XJ):: IF XJ THE
N 920
1070 IF YJ=0 THEN 1060
1080 CALL GCHAR(H(X+8),H(Y+2*YJ),TEK)::
IF TEK=33 THEN 1060
1090 GOTO 660
1100 !************************
1110 CALL CHAR(33,"FBFB007700DD0077",42,
"0810081008100810",120,"0306060504030205
0A050A050A050201E0B0B0D010E020D028D028D0
28D020C")
1120 CALL COLOR(1,15,2,2,16,2):: RESTORE
 1280 :: RETURN
1130 CALL CHAR(33,"FE0670777707E0EE",42,
"1000100008001000",120,"3748555B2F5B5F2D
5FAB9E6D152A241830C828E6D26AF4ECB2FAF46E
61916A04")
1140 CALL COLOR(1,11,2,2,16,2):: RESTORE
 1370 :: RETURN
1150 CALL CHAR(33,"FF818181FFAE2420",42,
"1028281008141408",120,"CF5FD24D47A7120F
020F0207F84020F080C04081121F20C000800000
F8102078")
1160 CALL COLOR(1,5,2,2,6,2):: RESTORE 1
460 :: RETURN
1170 CALL CHAR(33,"F7F7F7007F7F7F",42,"1
1,2)
1180 CALL CHAR(33,"F7F7F7007F7F7F",42,"1
008040804081020",120,"020A0A68776FE90DEE
0FF4031801191980A0A80EEEF697B077F02FC018
809898")
1190 CALL CHAR(120,"020A0A68776FE90DEE0F
F4031801191980A0A80EEEF697B077F02FC01880
9898"):: RESTORE 1550 :: RETURN
1200 GOSUB 1190
1210 CALL HCHAR(1,1,100,256):: CALL HCHA
R(9,1,105,512):: CALL CHAR(33,""):: DISP
LAY AT(2,14)SIZE(13):"'adddddddddbf" ::
DISPLAY AT(3,13)SIZE(14):"'hhadddddddddcg
"
1220 FOR T=2 TO 6 :: DISPLAY AT(T+2,14-T
)SIZE(2*T+2):"'"&RPT$("h",T*2)&"a" :: NE
XT T :: FOR T=11 TO 22 :: CALL HCHAR(T,4
,32,27):: NEXT T
1230 DISPLAY AT(13,4)SIZE(23):""" :F:QWp
ZQrtWsv  :trQ """ :: DISPLAY AT(14,4)SIZ
E(23):"# ;J;FXqFFFuX w  ;uFu #"
1240 DISPLAY AT(16,4)SIZE(23):"&'  BY RE
MCO POLMAN  &'" :: DISPLAY AT(17,13)SIZE
(5):"@1986" :: DISPLAY AT(20,9)SIZE(13):
"PRESS  BUTTON"
1250 DISPLAY AT(19,8)SIZE(15):USING "HIS
CORE #######":HI :: CALL SPRITE(#1,120,5
,49,121)
1260 CALL KEY(2,K,S):: IF S=0 THEN 1260
ELSE 470
1270 !************************
1280 DATA "  ,                +
          +                !!!!!!!!!!!(!
!!!+!!!!!!!!!!!"
1290 DATA ")   +       (!+  + (       )  *
   +       (!+  + (       *  *!!!+!!!!!!(!
+!!+!(!!!!!!*!!"
1300 DATA "*           (!+     (!(   !*  *
          (!+     (!(   !*  *!!!!!!!!!!(!
+!!!!(!(!!!!*!!"
1310 DATA "* !)        (       (!(!+  * (*
 !*        (       (!(!+  * (!!!*!!!!!!!!!
!!!!!!!(!+!!!!("
1320 DATA "   *  ( + + (  ! )  ( +     (
  *  ( + + (  ! *  ( +     (!!!*!!(!+!+!(
!!!!*!!!!+!!!!("
1330 DATA " )!*  (  !+!(     *     !   (
*!*  (  !+!(     *     !   (!*!!!!(!!!+!(
!!!!*!!!!!!!!!("
1340 DATA " *    (  ! ((!   *   +     (
*    (  ! ((!   *   +     (!*!!!!!!!!!(!
!!!!!!!!+!!!!!("
1350 DATA " *          (                (
*          (                (!*!!!!!!!!!!!
!!!!!!!!!!!!!!!"
1360 !************************
1370 DATA "!*!   (          +     ( +   !
*!   (          +     ( +   !*!!!!(!!!!!!(
!!!+!!!!!(!+!!!"
1380 DATA " *    (!)    (   +!(   ( + )
*    (!*   (   +!(   ( + * !!!!!!!!*!!!(
!!!+!(!!!!!+!*!"
1390 DATA "(        *   (   + (     + * (
      *   (   + (     + * (!!!!!!!*!!!!
!!!+!(!!!!!+!!!"
1400 DATA "(     ) *    ) (  ( (       (
```

```
     * *    * (   ( (        (!!!!!*!*!!!!
*!(!!!!(!!!!!!!"
1410 DATA "( )    * *   + *!(     (     )   (
 *    * *   + *!(     (     *   !!*!!!*!!!!+!
!!(!!!!(!!!!*!!"
1420 DATA "  *    *     +    (     (   + *
 *    *     +    (     (  + *   !!*!!!*!!!!+!
!!(!!!!!!!+!*!!"
1430 DATA " )*    *        !(   +       *
**    *        !(   +       *   !*!!!!!!!!!!!
!!(!!+!!!!!!*!!"
1440 DATA " *               (           *
*             (           *   !*!!!!!!!!!!!
!!!!!!!!!!!!!!!"
1450 !**********************
1460 DATA " *   ++           +     )
*   ++           +     *       !*!!++!!!(!!!
!!!+!!!!*!!!!!!"
1470 DATA " *!   +    (     ( +     *   !  )
*!   +    (     ( +     *   ! * !*!!!+!!!(!!!
!(!+!!!!*!!!!*!"
1480 DATA "(*    +    (!    ( +     *     * (
*    +    (!    ( +     *     * (!!!!+!!!!!!!
!(!+!!!!!!!!!*!"
1490 DATA "(   ++      +    ( +++    (    * (
  ++      +    ( +++    (    * !!!++!!!!!+!!
!!!+++!!!(!!!!!"
1500 DATA "     +    ) +   ! + +    ( )   (
   +    * +    ! + +    ( *   (!!!!+!!!*!+!!
!!!+!+!!!!!*!!("
1510 DATA "     +    *      ! +    (    *   (
   +    *      ! +    (    *   (!!!!+!!!*!!!!
!!!+!!!(!!!!!!("
1520 DATA " (        *    +    .  ) (        (
(        *    +      * (        (!(!!!!!!!*!!!+
!!!!!*!!!!!!!!("
1530 DATA " (        *           *          (
(        *           *          (!(!!!!!!!!!!!
!!!!!!!!!!!!!!!"
1540 !**********************
1550 DATA "!*""   )            +        (   )
!*#   *           +        (   * !*!!!*!!!!!(
!!!!+!!!!!!(!!*!"
1560 DATA "!*    *!) (!(!!!!+!!!)   ( !*!!
*    *!* (!(!&'!+!""!*   ( !*!!*!!!!!*!(!(
!!!!+!#!*!!(!!*!"
1570 DATA " *      *!( ( )        *   (   *
*      *!( ( *       *   (   * !(!!!!!*!!!!!
*!!!!!!!!!(!!!!"
1580 DATA "!(!)   !*!!""!!*!(    !    (!(
!(!*   !*!!#!!*!(    !    (!(   !(!*!!!*!!!!
!*!(!!!!!!!!!(!!"
1590 DATA "!( *!( *!   + * (!!+        (   !
( *!( *!   + * (!!+        (   !(!!!(!!!!!+!
!!(!!+!!!!!!(!!"
1600 DATA " (    (!!""!!     (!! )!!!   ()!
 (    (!!#!!     (!! *!!!   (*!!!!!!!!!!!!!
!!!(!!!*!!!!!!*!"
1610 DATA "!""""""!xz!&'!""!!!((!!*    (
 *!!###!y(!!!!#!!!((!!*    (   *!!!!!!!!!!
!!!!!!!(!!!!!!(!!!!"
1620 DATA " (                (        (
(                (        (     !(!!!!!!!!!!!
!!!!!!!!!!!!!!!"
1630 !**********************
1640 CALL MOTION(#1,0,0):: M=M-1 :: CALL
 PATTERN(#1,60):: CALL PATTERN(#1,92)::
FOR T=220 TO 110 STEP -5 :: CALL SOUND(-
50,T,0):: NEXT T :: GOSUB 1830 :: IF M>0
 THEN 800
1650 FOR T=10 TO 16 :: CALL HCHAR(T,11,3
1,13):: NEXT T :: DISPLAY AT(11,11)SIZE(
9):"GAME OVER" :: DISPLAY AT(13,9)SIZE(1
3):"PRESS  BUTTON" :: DISPLAY AT(15,11)S
IZE(9):"TO REPLAY"
1660 CALL KEY(2,K,S):: IF S<1 THEN 1660
ELSE 460
1670 IF D(SCR)>0 THEN DISPLAY AT(1,8)SIZ
E(14):"OPEN EACH+DOOR" :: CALL SOUND(-20
0,220,0):: DISPLAY AT(1,8)SIZE(14):"
     +     " :: GOTO 1060
1680 CALL MOTION(#1,127,0):: CALL SOUND(
-100,880,0,990,0):: SCR=SCR+1 :: CALL DE
LSPRITE(ALL):: CALL VCHAR(1,3,31,672)::
IF SCR<5 THEN 490
1690 CALL VCHAR(1,3,33,672):: FOR T=7 TO
 15 :: CALL HCHAR(T,T-3,32,3):: NEXT T :
: CALL VCHAR(1,4,40,8)
1700 CALL HCHAR(14,14,32,2):: CALL HCHAR
(15,15,32,4):: FOR T=9 TO 14 :: CALL HCH
AR(T,16,32,9):: NEXT T :: FOR T=17 TO 23
 STEP 2
1710 CALL HCHAR(10,T,34):: CALL HCHAR(11
,T,35):: NEXT T :: CALL SPRITE(#2,120,5,
97,153,#1,124,11,1,25)
1720 FOR T=1 TO 49 STEP 3 :: CALL LOCATE
(#1,T,25):: FOR X=1 TO 15 :: NEXT X :: C
ALL SOUND(1,880,0,990,0):: NEXT T :: CAL
L PATTERN(#1,128):: FOR T=49 TO 105 STEP
 8 :: CALL LOCATE(#1,T,T-16)
1730 FOR X=1 TO 15 :: NEXT X :: CALL SOU
ND(1,-5,0):: NEXT T :: FOR T=97 TO 137 S
TEP 8 :: Q=-Q :: CALL PATTERN(#1,2*Q+130
):: CALL LOCATE(#1,105,T):: FOR X=1 TO 2
0 :: NEXT X
1740 CALL SOUND(1,-5,0):: NEXT T :: FOR
T=1 TO 20 :: CALL LOCATE(#1,100,137):: F
OR X=1 TO 10 :: NEXT X :: CALL LOCATE(#1
,105,137):: CALL SOUND(1,-6,0):: FOR X=1
 TO 20 :: NEXT X :: NEXT T
1750 CALL DELSPRITE(ALL):: M=M+1 :: LV=L
V+1 :: SCR=1 :: P=P+100*LV :: CALL VCHAR
(1,3,31,672):: GOTO 480
1760 GOSUB 1810
1770 FOR T=1 TO 8 :: READ A$ :: DISPLAY
AT(T*3-2,1):A$ :: NEXT T :: GOSUB 1820 :
: GOSUB 1830 :: CALL HCHAR(1,30,44)
1780 X=177 :: Y=25 :: CALL SPRITE(#1,124
,11,X,Y):: FOR T=1 TO 6
1790 Q=-Q :: S=(RND*5+3*LV)*Q :: CALL SP
RITE(#T*2,120,RND*13+3,T*24+1,1,0,S):: I
F LV>2 THEN CALL SPRITE(#T*2+1,120,RND*1
3+3,T*24+1,125,0,S)
1800 NEXT T :: GOTO 910
1810 CALL SOUND(-100,880,0,990,0):: CALL
 DELSPRITE(ALL):: CALL VCHAR(1,3,31,672)
:: DISPLAY AT(12,8)SIZE(16):"BACK TO SCR
EEN";SCR :: ON SCR GOSUB 1110,1130,1150,
1170 :: GOTO 1770
1820 A$=STR$(P):: DISPLAY AT(24,6)SIZE(L
EN(A$)):A$ :: RETURN
1830 CALL HCHAR(24,16,33,4):: CALL HCHAR
(24,16,128,M):: RETURN
```

```
<KP> 500 DATA 252,252,252,124,156,164,16
9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1369
<GN> 510 DATA 63,63,63,63,61,54,26,106,6
1,54,26,106,106,154,166,169,0,0,0,0,0,0,
0,1341
<CF> 520 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0
<CG> 530 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0
<CH> 540 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0
<CI> 550 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0
```

Ende des Listings

```
type top.txt
                    Achtung!

Auch Kassettenbenutzer koennen dieses Programm verwenden, allerdings muessen
sie auf die Higscore-Liste verzichten. Folgende aenderungen sind notwendig
damit 'The Wipping Top' auch auf Kassette lauffaehig ist:

Teil 1
 180 CLOAD


Teil 2
 960 OPEN #1,4,0,"C:TOP.SET"
 1210 REM
 1220 GOTO 580
 3300 K=1:OP=1:R=2
 3310 TRAP 3320:OP=OP+12:GOTO 3310
 3320 GRAPHICS 0:GOSUB 410:DL=PEEK(560)+256*PEEK(561):POKE DL+3,70
 3370 GOTO 3320


Teil 3
 10 GRAPHICS 0:TRAP 70:OPEN #1,8,0,"C:TOP.SET"
```

CRAZY COWBOYS

Westernhelden ohne Hemd und Hose!

Peng

Revolverhelden einmal ganz anders! Aus einem harten Schießduell wird ein lustiges Spielchen für zwei tollkühne Schützen. Anstatt einander umzulegen, versuchen zwei „verrückte" Cowboys, sich gegenseitig die Kleidungsstücke wegzuschiessen.

Peng

Nach Abspielen der Melodie 'Von den blauen Bergen' und der Einblendung einiger provozierender Sprüche (auf Englisch) stehen sich zwei Revolverhelden in der Wüste Nevadas gegenüber. Setzen Sie sich nun mit Ihrem auserwählten Gegner und den zwei Joysticks an den Bildschirm! Die „Kanonen" immer im Anschlag, können Sie mittels Joystick Ihren Helden auf und ab bewegen und „blaue Bohnen" in unbegrenzter Zahl auf Ihn feuern. Die einzige Deckung wird Ihnen von einem Planwagen geboten, der ständig zwischen den beiden Streithähnen durch die Wüste rollt. Die verstreut herumstehenden Kakteen bieten leider keinen Schutz!

Allerdings erzielen Sie mit Ihren Schüssen nur dann eine Wirkung, wenn Sie den Oberkörper des Gegners treffen (vom Gürtel bis zur Hutspitze). Bei den ersten sechs Treffern dieser Art fliegt jeweils der Hut weg, der sogleich durch einen neuen ersetzt wird. Sichtlich erschrocken wirft der Getroffene die Arme hoch. Doch erst beim

nächsten Treffer wird's ernst, denn da verliert er die Hose!
Während nun der Gewinner seiner Schadenfreude Ausdruck gibt, steht der Verlierer in der Unterhose in der weiten Wüste Nevadas da. Verschämt zieht er noch sein Hemd runter und stößt einen derben Fluch aus (in English!).

Trotz des relativ langsamen TI-EX-Basic sind die Cowboys so schnell und beweglich, daß es gar nicht so einfach ist, einen Treffer zu landen. Dafür sorgt das möglichst kurz gehaltene Hauptprogramm. Beiden Spielern ist es allerdings nicht möglich, gleichzeitig zu feuern, sondern immer nur abwechselnd.
Wer möchte, kann das Spiel auch in 'halbierter' Größe ablaufen lassen. Dazu ist lediglich Zeile 780 in CALL MAGNIFY (3) zu ändern.

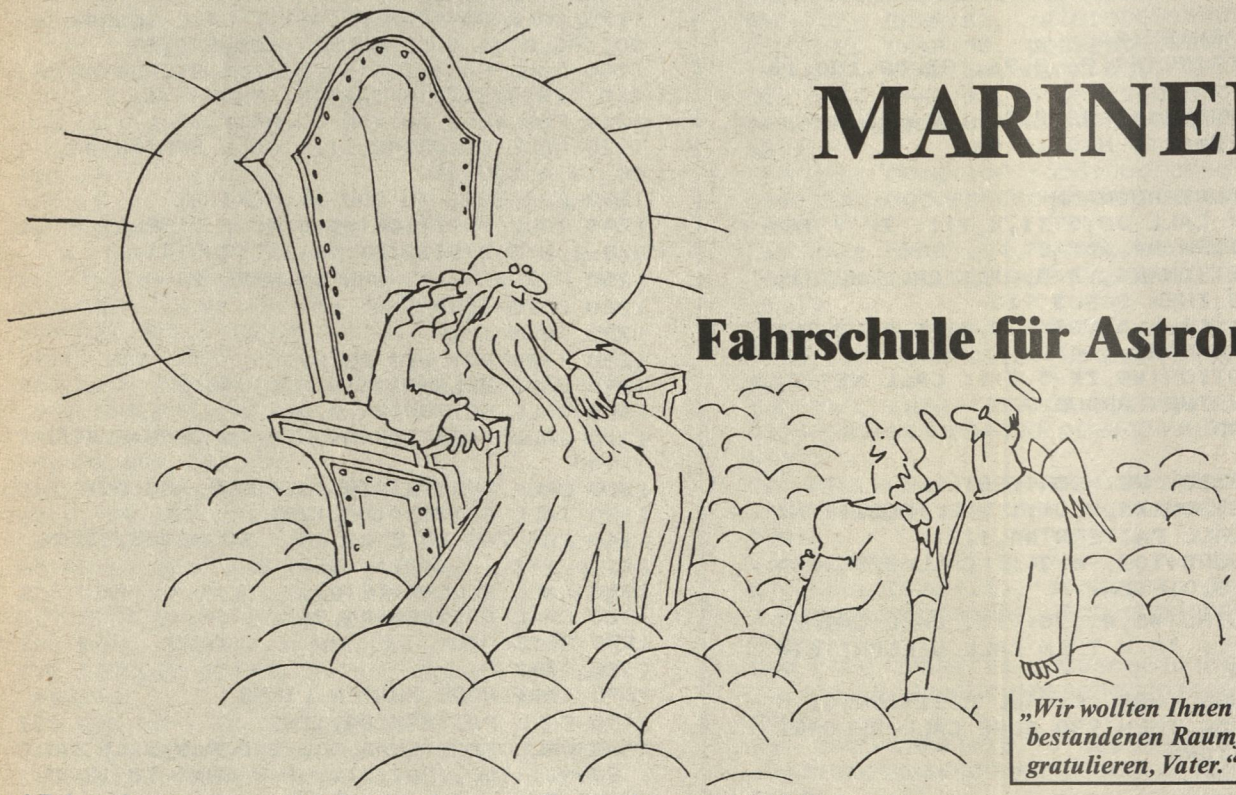
Hinweis:
Nicht vergessen, die Alpha-Lock-Taste zu lassen, da sonst die Joystick-Steuerung nach oben blockiert ist!

```
100 !**********************
110 !*                    *
120 !*     C R A Z Y      *
130 !*                    *
140 !*   C O W B O Y S    *
150 !*                    *    !*   ein
 spassiges     *    !* actionspiel fuer
   *    !*  zwei joysticks     *
160 !* in ti ext.basic     *    !*
                *
170 !*   8674 bytes        *    !*
                *
180 !*      (c)1986        *
190 !*         by          *    !*  mich
ael oswald     *    !*
   *
200 !**********************
210 CALL CLEAR :: CALL SCREEN(2)
220 FOR SET=0 TO 12 :: CALL COLOR(SET,16
,2):: NEXT SET
230 DATA 250,494,250,494,500,659,500,659
,250,554,250,494
240 DATA 250,415,250,494,500,659,750,400
00,250,659,250,740
250 DATA 500,831,500,831,250,988,250,988
,250,988,250,831,500,740
260 DIM AR(38):: FOR SD=1 TO 38 :: READ
AR(SD):: NEXT SD
270 DISPLAY AT(1,10):"C R A Z Y" :: DISP
LAY AT(4,8):"C O W B O Y S"
280 CALL HCHAR(7,7,ASC("*"),19)
290 DISPLAY AT(24,3):"(c)1986 by michael
 oswald"
300 FOR SD=1 TO 37 STEP 2
310 CALL SOUND(AR(SD),AR(SD+1),5,AR(SD+1
)-305,13,AR(SD+1)+2500,7)
320 NEXT SD
330 FOR BR=1 TO 300 :: NEXT BR
340 DISPLAY AT(10,1):"""hey mac, i bet y
ou're the  lousiest gunman in the wholew
est."""
350 FOR BR=1 TO 900 :: NEXT BR
360 DISPLAY AT(14,1):"""you bloody basta
rd, i can  shoot your pants down any   t
ime i'd like to !"""
370 FOR BR=1 TO 900 :: NEXT BR
380 DISPLAY AT(18,1):"""har,har, let's s
ee about   that, you son of a bitch !"""
390 FOR BR=1 TO 900 :: NEXT BR
400 DISPLAY AT(12,12)ERASE ALL:"READY ?"
410 CALL CHAR(36,"01030000000000000000000
000000000080C00000000000000")
420 CALL CHAR(40,"00000000000000000000101
000000000000000000000000000C0C")
430 CALL CHAR(44,"00000000000000000000101
010000000000000000000000000C0C04")
440 CALL CHAR(48,"00000000000000000000101
010101000000000000000000000C0C040404")
450 CALL CHAR(92,"03070F1C1818180804171 7
171F171010C0E0F0381818181020E8E8E8F8E808
08")
460 CALL CHAR(56,"00000000010000000000000
000000000000000000000000000000")
470 CALL CHAR(52,"181818D8D8DBDBDB")
480 CALL CHAR(53,"DBDBDBDBDBDBDBDB")
490 CALL CHAR(54,"DBFBFB1B1F1F1818")
500 CALL CHAR(96,"01030100070505050201 03
020202060080C08080F0D0D0D020C0C040404040
60")
510 CALL CHAR(100,"00010000070505050201 0
30202020600C0E0C080F0D0D0D020C0C04040404
060")
520 CALL CHAR(104,"01030100070505050201 0
101010101380C08080F0D0D0D020C0E02020203
000")
530 CALL CHAR(108,"00010000070505050201 0
10101010103C0E0C080F0D0D0D020C0E02020203
000")
540 CALL CHAR(112,"010301301F01010102010
302020206008 0C08080F0D0D0D020C0C04040404
060")
550 CALL CHAR(116,"00010000070505050201 0
10101010103C0E0C086FCC0C0C020C0E02020203
000")
560 DISPLAY AT(12,12):" G O  !"
570 CALL CHAR(120,"00040404070101010201 0
3020202060000D0D090F0C0C0C020C0C04040404
060")
580 CALL CHAR(124,"00050504070101010201 0
101010101030090909 0F0C0C0C020C0E02020203
000")
590 CALL CHAR(128,"010301000F09091912010
101010101380C08080F8C8C8CC24C0C04040404
060")
600 CALL CHAR(132,"000100000F09091912010
10101010103C0E0C080F8C8C8CC24C0C04040404
060")
610 CALL CHAR(136,"01030100000305050505 0
300000001 0380C0808080E0D0D0D0D0E00000004
060")
620 CALL CHAR(140,"00010000000305050505 0
30000000103C0E0C08080E0D0D0D0D0E00000004
060")
630 A$(1)="HAHA" :: A$(2)="HOHO" :: A$(3
)="HIHI" :: A$(4)="SEXY" :: A$(5)="NICE"
 :: A$(6)="CHIC"
640 B$(1)="OH SHAME" :: B$(2)="DAMMIT"  :
: B$(3)="GODDAM" :: B$(4)="BULLSHIT"
650 !***
660 !*** INITIALISIERUNG
670 FOR SET=0 TO 12 :: CALL COLOR(SET,2,
1):: NEXT SET
680 CALL CLEAR :: CALL SCREEN(12)
690 CALL COLOR(3,4,1)
```

```
700 CALL HCHAR(1,3,52):: CALL HCHAR(2,3,
53):: CALL HCHAR(3,3,54)
710 CALL HCHAR(17,21,52):: CALL HCHAR(18
,21,53):: CALL HCHAR(19,21,54)
720 CALL HCHAR(8,9,52):: CALL HCHAR(9,9,
54)
730 CALL HCHAR(4,25,52):: CALL HCHAR(5,2
5,54)
740 CALL HCHAR(16,5,52):: CALL HCHAR(17,
5,54)
750 CALL HCHAR(11,27,52):: CALL HCHAR(12
,27,54)
760 CALL HCHAR(22,31,52):: CALL HCHAR(23
,31,54)
770 CALL HCHAR(23,12,52):: CALL HCHAR(24
,12,54)
780 CALL MAGNIFY(4)
790 CALL SPRITE(#4,96,7,96,192,#8,100,13
,96,60)
800 CALL SPRITE(#5,92,2,96,128,-10,0)
810 E=4 :: D=4 :: H=0 :: G=0
820 !***
830 !*** HAUPTPROGRAMM
840 E=-E :: CALL JOYST(1,X,Y):: IF Y THE
N CALL PATTERN(#4,100+E)
850 CALL MOTION(#4,Y*-5,0):: CALL KEY(1,
Q,S):: IF S THEN GOSUB 910
860 D=-D :: CALL JOYST(2,W,Z):: IF Z THE
N CALL PATTERN(#8,104+D)
870 CALL MOTION(#8,Z*-5,0):: CALL KEY(2,
Q,U):: IF U THEN GOSUB 970
880 GOTO 840
890 !***
900 !*** COWBOY RE. SCHIESST
910 CALL MOTION(#4,0,0):: CALL POSITION(
#4,R,C):: CALL PATTERN(#4,112)
920 CALL SOUND(100,-6,0):: CALL SPRITE(#
1,56,2,R,C+3,0,-126)
930 CALL COINC(#5,#1,20,V):: CALL COINC(
#8,#1,12,T):: IF V THEN CALL DELSPRITE(#
1):: RETURN
940 IF T THEN 1030 :: CALL POSITION(#1,M
,N):: IF N>50 THEN 930 ELSE CALL DELSPRI
TE(#1):: RETURN
950 !***
960 !*** COWBOY LI. SCHIESST
970 CALL MOTION(#8,0,0):: CALL POSITION(
#8,I,J):: CALL PATTERN(#8,116)
980 CALL SOUND(100,-6,0):: CALL SPRITE(#
1,56,2,I,J-2,0,126)
990 CALL COINC(#5,#1,20,V):: CALL COINC(
#4,#1,12,T):: IF V THEN CALL DELSPRITE(#
1):: RETURN
1000 IF T THEN 1290 :: CALL POSITION(#1,
M,N):: IF N<200 THEN 990 ELSE CALL DELSP
RITE(#1):: RETURN
1010 !***
1020 !*** HUT WEG LINKS
1030 CALL DELSPRITE(#1):: H=H+1
1040 CALL MOTION(#8,0,0)
1050 CALL POSITION(#8,A,B):: IF H=6 THEN
 1140
1060 CALL SPRITE(#10,36,13,A,B,-20,-50)
1070 CALL PATTERN(#8,124)
1080 FOR Z=3 TO 5 :: CALL SOUND(-5,2500+
Z*100,2*Z,-7,6*Z):: NEXT Z
1090 CALL DELSPRITE(#10)
1100 CALL PATTERN(#8,100)
1110 GOTO 860
1120 !***
1130 !*** HOSE RUNTER LINKS
1140 CALL PATTERN(#8,132)
1150 CALL SPRITE(#3,40,16,A,B,0,0):: CAL
L SOUND(-100,1000,3):: FOR BR=1 TO 12 ::
 NEXT BR
1160 CALL SOUND(-100,900,3):: FOR BR=1 T
O 12 :: NEXT BR
1170 CALL PATTERN(#3,44):: CALL SOUND(-1
00,800,3):: FOR BR=1 TO 12 :: NEXT BR
1180 CALL SOUND(-100,700,3):: FOR BR=1 T
O 12 :: NEXT BR
1190 CALL PATTERN(#3,48):: CALL SOUND(-1
00,600,3):: CALL SOUND(-100,500,3)
1200 CALL POSITION(#4,J,K):: DISPLAY AT(
J/8,K/8-4)SIZE(4):A$(INT(RND*6)+1)
1210 FOR BR=1 TO 150 :: NEXT BR
1220 CALL COLOR(#8,1):: CALL SPRITE(#2,1
40,13,A,B,0,0)
1230 FOR BR=1 TO 200 :: NEXT BR
1240 CALL POSITION(#8,V,W):: DISPLAY AT(
V/8-1,W/8-3)SIZE(8):B$(INT(RND*4)+1)
1250 FOR BR=1 TO 800 :: NEXT BR
1260 GOTO 1590
1270 !***
1280 !*** HUT WEG RECHTS
1290 CALL DELSPRITE(#1):: G=G+1
1300 CALL MOTION(#4,0,0)
1310 CALL POSITION(#4,0,P):: IF G=6 THEN
 1400
1320 CALL SPRITE(#10,36,7,0,P,-20,50)
1330 CALL PATTERN(#4,120)
1340 FOR Z=3 TO 5 :: CALL SOUND(-5,2500+
Z*100,2*Z,-7,6*Z):: NEXT Z
1350 CALL DELSPRITE(#10)
1360 CALL PATTERN(#4,96)
1370 GOTO 840
1380 !***
1390 !*** HOSE RUNTER LINKS
1400 CALL PATTERN(#4,128)
1410 CALL SPRITE(#3,40,16,0,P,0,0):: CAL
L SOUND(-100,1000,3):: FOR BR=1 TO 12 ::
 NEXT BR
1420 CALL SOUND(-100,900,3):: FOR BR=1 T
O 12 :: NEXT BR
1430 CALL PATTERN(#3,44):: CALL SOUND(-1
00,800,3):: FOR BR=1 TO 12 :: NEXT BR
1440 CALL SOUND(-100,700,3):: FOR BR=1 T
O 12 :: NEXT BR
1450 CALL PATTERN(#3,48):: CALL SOUND(-1
00,600,3):: CALL SOUND(-100,500,3)
1460 CALL POSITION(#8,V,W):: DISPLAY AT(
V/8,W/8+2)SIZE(4):A$(INT(RND*6)+1)
1470 FOR BR=1 TO 150 :: NEXT BR
1480 CALL COLOR(#4,1):: CALL SPRITE(#2,1
36,7,0,P,0,0)
1490 FOR BR=1 TO 200 :: NEXT BR
1500 CALL POSITION(#4,J,K):: DISPLAY AT(
J/8-1,K/8-3)SIZE(8):B$(INT(RND*4)+1)
1510 FOR BR=1 TO 800 :: NEXT BR
1520 GOTO 1590
1530 CALL DELSPRITE(#3,#2)
1540 CALL PATTERN(#4,96):: CALL COLOR(#4
```

```
,7)
1550 G=0
1560 GOTO 840
1570 !***
1580 !*** SCHLUSSBILD
1590 CALL DELSPRITE(ALL)
1600 CALL CLEAR :: CALL SCREEN(2):: CALL
 COLOR(5,16,2,6,16,2,7,16,2,8,16,2)
1610 DISPLAY AT(6,8):"IF YOU"
1620 DISPLAY AT(8,8)BEEP:"WANNA SHOOT AG
AIN"
1630 DISPLAY AT(13,8):"PRESS Y FOR YES"
1640 DISPLAY AT(16,8):"PRESS N FOR NO"
1650 CALL KEY(0,KE,ST)::
1660 IF ST=0 THEN 1650 :: IF KE=121 OR K
E=89 THEN 1670 ELSE CALL CLEAR :: CALL D
ELSPRITE(ALL):: END
1670 CALL CLEAR :: CALL SCREEN(12):: CAL
L COLOR(5,2,1,6,2,1,7,2,1,8,2,1):: GOTO
690
```

Ende des Listings



MARINER V

Fahrschule für Astronauten

„Wir wollten Ihnen nur zur bestandenen Raumfahrtprüfung gratulieren, Vater."

Drei Raketenbasen erwarten Ihre Landung! Sollten Sie es schaffen, unbeschadet aufzusetzen, müssen Sie ebenso einwandfrei wieder starten.

Leider gibt es eine Schwierigkeit beim Versuch, wieder in den freien Raum zu gelangen. Umherfliegende Meteore verhindern das leichte Hoch- und Runterfliegen. Auf vier verschiedenen Bahnen kreisen diese um den Planeten. Jedoch können Sie beim Wiederaufstieg die Steinchen abschießen. Jeder Treffer bringt Ihnen einen Punkt ein. Eine ordentliche Landung sogar zehn Punkte. Jede der drei Basen muß pro Durchgang dreimal angeflogen werden, wobei sich die Landeplattform nach jeder erfolgreichen Landung verkleinert. Der Zustand der Plattform wird durch die darunter angeordneten Fähnchen angezeigt.

Seine Rakete bewegt man per Joystick. Der Feuerknopf dient zum Starten und Laden, sowie Feuern. Insgesamt bleiben dem Spieler zwei Reserveraketen.

Zum Programm:

Das Programm besteht aus drei Teilen:

MARINERLOADER poket den Maschinenteil in den Hauptspeicher und sichert Ihnen anschließend in der Binärdatei MARINER.OBJ auf Diskette/Kassette. Ein möglicher Tippfehler in den DATA-Zeilen wird durch Prüfsummenbildung erkannt.

PIC.MAKER generiert das Titelbild und schreibt es dann in der Binärdatei MARINER.PIC auf Diskette /Kassette.

MARINER V lädt MARINER.OBJ sowie MARINER.PIC und steuert den Spielablauf.

Um also ein lauffähiges MARINER V-Programm zu erhalten, müssen zuerst MARINERLOADER und PIC.MAKER gestartet werden.

Teil 1

```
100  REM  *** DATA - POKER FUER
     ARINER V ***
110  REM
120  LOMEM: 8 * 4096
130 ST = 6 * 4096
140 EN = ST + 5068
150 ZE = 340: HOME
160  FOR AD = ST TO EN STEP  14
170 SU = 0
180  FOR DA = AD TO AD + 13
190  READ PO
200  POKE DA,PO
210 SU = SU + PO
220  NEXT DA
230  READ CH
240  IF SU <  > CH THEN 300
250 ZE = ZE + 10
260  VTAB 12: HTAB 18: PRINT  ZE
270  NEXT AD
280  PRINT  CHR$ (4)"BSAVE  MARI
     R.OBJ,A$6000,L$13D4"
290  END
```

RUBBERBALL

TI-99/4a + Extended Basic

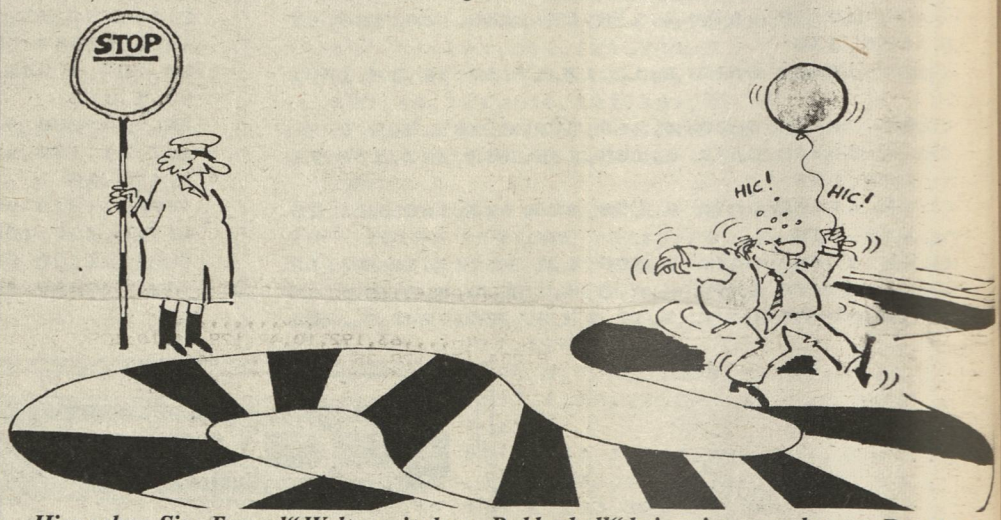
Ein kleiner hüpfender Gummiball sammelt für Sie wichtige Dinge ein. Steuern Sie ihn geschickt durch die vielfältigen Gefahrenquellen!!!

Der kleine Rubberball kann rollen, hüpfen, springen und sich durch den Boden graben. Aber Vorsicht!!! RUBBERBALL wird von verschiedenen Gefahren bedroht! So können bestimmte Steine unter Rubberball nachgeben und ihn in die tiefer gelegene Spielebene befördern (ist nicht unbedingt gefährlich!).
Es gibt allerdings auch Ziegelsteine, die unter RUBBERBALL explodieren und ihn ins Jenseits befördern. Solange sich RUBBERBALL jedoch bewegt, können ihm diese Gefahren nichts anhaben. Erst wenn er auf den Backsteinen ruht, treten die eingebauten Fallen in Aktion.
Außerdem sehen Sie auf dem Spielfeld sehr gefährliche Nagelbretter. Sollte RUBBERBALL auf ein solches Feld gelangen, wird ihm ganz schnell die Luft ausgehen....
Auch wenn RUBBERBALL zu große Sprünge wagt, muß er sich aus dem Spiel verabschieden, da er dann zerplatzt.

Steuerung:

Die Steuerung des RUBBERBALLS erfolgt mit den Tasten: S = links und D = rechts. Wenn Sie eine dieser Tasten betätigen, während der Ball schon nach oben springt, wird er einen diagonalen Sprung in die jeweilige Richtung ausführen.
Als Hilfsmittel befindet sich in jedem Spielfeld noch ein Teleporter (zu erkennen an der "T"-Form!), der aber nicht immer zu einer günstigen Stelle teleportiert.
Haben Sie in einem Spiel alle Schlüssel eingesammelt, gelangen Sie in das nächste Bild. Befinden Sie sich einmal in einer vertrackten Situation, aus der Sie nicht mehr herauskommen und außerdem keine Möglichkeit haben eines Ihrer RUBBER-Leben zu opfern, können Sie mit FCTN "BEGIN" das Spiel abbrechen und neu starten. Das Spiel ist beendet, wenn Sie entweder alle Schlüssel der vorhandenen Spielfelder eingesammelt oder wenn Sie Ihre 10 RUBBER-Leben verloren haben. Danach wird dann Ihr Score und der Highscore angezeigt.

Also RUBBERfans – ran an den TI!!!



Hier sehen Sie „Freund" Walter mit dem „Rubberball" beim einsammeln von Punkten.

●●● Teil 1 ●●●

```
10 CALL CLEAR
20 CALL CHARPAT(66,B$,65,A$,76,L$,82,R$,
85,U$,69,E$,84,T$,83,S$,79,O$,70,F$,80,P
$,78,N$)
30 CALL CHARPAT(46,POI$)
40 CALL CHAR(81,B$,82,A$,83,L$,84,R$,85,
U$,86,E$,87,POI$,88,T$,89,S$,90,O$,91,F$
,92,P$,93,N$)
50 CALL SCREEN(2)
60 CALL COLOR(2,2,1)
70 RESTORE 80 :: FOR I=1 TO 93 :: READ Z
A,CH$ :: CALL CHAR(ZA,CH$):: NEXT I
80 DATA 97,000000000001071C
90 DATA 98,03070C1870C0
100 DATA 99,FFC0
110 DATA 100,FF01
120 DATA 101,10E0781D060201
130 DATA 102,0000F887
140 DATA 103,00000000C0202060
150 DATA 104,0000000101030301
160 DATA 105,3060C08080000080
170 DATA 106,000001060C101020
180 DATA 107,00FF0000000F1F70
190 DATA 108,00F3040201E01088
200 DATA 109,0000000000808083
210 DATA 110,40413E0000070898
220 DATA 64,B0B0303030F05878
230 DATA 111,010101
240 DATA 112,808080C0C0C06030
250 DATA 113,2040404040404020
260 DATA 114,50502F100F
```

RUBBERBALL

```
270 DATA 115,88880810E0000102
280 DATA 116,84888899998807
290 DATA 117,6E1212AA56B8C7
300 DATA 118,3838382CA616060C
310 DATA 119,180C0C0C0C060303
320 DATA 120,100C030000008040
330 DATA 121,0000FF
340 DATA 122,0438C00000000F10
350 DATA 123,000000000000C03F
360 DATA 124,00000001010F1EE6
370 DATA 125,1830C08080
380 DATA 126,0303030306060607
390 DATA 127,38040380403807
400 DATA 128,000080403F01837F
410 DATA 129,100F0000FEFF01
420 DATA 130,00FF000003FFFD01
430 DATA 131,0EFC1C7FFFF8E083
440 DATA 132,007FFF8001007FFF
450 DATA 133,00FCFE830383FEFC
460 DATA 134,0C0E0D0C1C336078
470 DATA 135,807F00FF00807F
480 DATA 136,03FF06FE0C7C9878
490 DATA 137,0
500 DATA 138,010303060C183060
510 DATA 139,070C0C0C0C0301
520 DATA 140,80000000000080C0
530 DATA 141,00FFFF030301
540 DATA 142,C7C03804E39CE3D8
550 DATA 143,00FF0001827C00FF
560 DATA 65,FF7F90080810200C0
570 DATA 66,FFFF838346262C5C
580 DATA 67,C08000000101
590 DATA 68,0018689020DF33E3
600 DATA 69,6060303018181818
610 DATA 70,0000000000070F1E
620 DATA 71,C4637866C1B00807
630 DATA 72,0000F80780700FFF
640 DATA 73,00030CF0071DEAF5
650 DATA 74,383870B060E0C080
660 DATA 75,0103070F0D1830C0
670 DATA 76,C381018180C0C0C0
680 DATA 77,18989898F0
690 DATA 78,0D06060381818181
700 DATA 79,807F0000E0FFE0E0
710 DATA 80,00FF00000000FF
720 DATA 36,7B860C183060C0C0
730 DATA 33,0101010303061C1F
740 DATA 34,808080C0E0F0F0F8
750 DATA 35,C060606030301818
760 DATA 37,C160303018181818
770 DATA 38,E0E0E0E07F60607F
780 DATA 39,01030306FC1830E0
790 DATA 40,800000000103070C
800 DATA 41,3060C0FF8000FF00
810 DATA 42,783C3CFC3C3CFE1E
820 DATA 43,06030301
830 DATA 44,00000080C0C06060
840 DATA 45,0C0C0C0606030301
850 DATA 46,301F191B1E
860 DATA 47,60C0800000010306
870 DATA 48,1F303F70CF80F00F
880 DATA 49,FF00FF00FF0001FE
890 DATA 50,FB79D818F818F818
900 DATA 51,0080C06070180C06
910 DATA 52,303018180C060603
920 CALL HCHAR(22,4,45)
930 CALL CHAR(53,"FFFFFFFFFFFFFFFFFFFF")
940 CALL VCHAR(1,3,53,24)
950 CALL HCHAR(1,3,53,29)
960 CALL VCHAR(1,31,53,24)
970 CALL HCHAR(24,3,53,29)
980 DATA 54,01
990 DATA 55,80C0C06060303030
1000 DATA 56,0C1E3160C0C0C0C0
1010 DATA 57,000000FF
1020 DATA 58,000102FC
1030 DATA 59,18F8181818181818
1040 DATA 60,060301
1050 DATA 61,030181C060303018
1060 DATA 62,008080C060303030
1070 DISPLAY AT(2,5):"VWXW YZ[X"
1080 DISPLAY AT(5,8):"\TVYVJXY"
1090 DISPLAY AT(16,12):"TUQQVT QRSS"
1100 RESTORE 1110 :: FOR I=1 TO 92 :: RE
AD A,B,C :: CALL HCHAR(A,B,C):: NEXT I
1110 DATA 12,5,97,12,6,98,12,7,99,12,8,1
00,12,9,101,12,10,102,12,11,103
1120 DATA 13,4,104,13,5,105,13,6,106,13,
7,107,13,8,108,13,9,109,13,10,110,13,11,
64
1130 DATA 14,4,111,14,5,112,14,6,113,14,
7,114,14,8,115,14,9,116,14,10,117,14,11,
118
1140 DATA 15,5,119,15,6,120,15,7,121,15,
8,122,15,9,123,15,10,124,15,11,125
1150 DATA 16,5,126,16,6,127,16,7,128,16,
8,129,16,9,130,16,10,131,16,11,132,16,12
,133
1160 DATA 17,5,134,17,6,135,17,7,136,17,
8,137,17,9,138,17,10,139,17,11,140
1170 DATA 18,4,141,18,5,142,18,6,143,18,
7,65,18,8,66,18,9,67,18,10,68,18,11,69
1180 DATA 19,4,70,19,5,71,19,6,72,19,7,7
3,19,8,74,19,9,75,19,10,76,19,11,77
1190 DATA 20,4,78,20,5,79,20,6,80,20,7,3
6,20,8,33,20,9,34,20,10,35
1200 DATA 21,4,37,21,5,38,21,6,39,21,7,4
0,21,8,41,21,9,42,21,10,43,21,11,44
1210 DATA 22,5,46,22,6,47,22,7,48,22,8,4
9,22,9,50,22,10,51,22,11,52
1220 DATA 23,4,54,23,5,55,23,6,56,23,7,5
7,23,8,58,23,9,59,23,10,60,23,11,61,23,1
2,62
1230 CALL SCREEN(8)
1240 CALL KEY(0,K,S):: IF S=0 THEN 1240
1250 CALL CLEAR :: CALL CHARSET
1260 CALL SCREEN(2):: FOR I=2 TO 14 :: C
ALL COLOR(I,16,5):: NEXT I
1270 CALL COLOR(13,10,1)
1280 CALL CHAR(97,"FDFDFD00DFDFDF",98,"F
DFDFD00DFDFDF",99,"FF8181E7E7E7E7FF",112
,"E0A0E0301A0F0602")
1290 CALL CHAR(104,"3C7EFFFF7E",105,"AAA
AAAAAAAAAFFFF",100,"DFDFDF00DFDFDF")
1300 CALL CHAR(129,"5ADBFFBDC3FFFF7E",13
0,"7EFFFFDBDBBDC37E",131,"7EEDEDFFDEE1FF
7E")
1310 CALL CHAR(132,"7EB7B7FF7B87FF7E",13
3,"7EFFDBDBFFC3BD7E")
1320 DISPLAY AT(1,4):"PRESS ANY KEY TO L
OAD"
1330 DISPLAY AT(5,1):"" :: DISPLAY AT(6,
```

```
1):"  R U
B B E R  B A L L   "
1340 DISPLAY AT(7,1):""
1350 DISPLAY AT(22,1):"(C) BY BERND HEBR
OCK"
1360 CALL KEY(0,K,S)::.IF S=0 THEN 1360
1370 ! NAECHSTE ZEILE ERST          EINGE
BEN WENN DAS            PROGRAMM UEBERPRU
EFT         IST
1380 RUN "CS1"
```

●●● Teil 2 ●●●

```
10 HI=0
20 CALL CLEAR :: CALL SCREEN(2)
30 DIM A$(22)
40 BILD=0 :: SCORE=0 :: MEN=10
50 RANDOMIZE
60 BILD=BILD+1 :: OK=0 :: CALL CLEAR ::
CALL DELSPRITE(ALL)
70 FA1=INT(RND*14)+1 :: FA2=INT(RND*14)+
1 :: IF FA2<3 THEN FA2=FA2+2 ELSE IF FA1
=FA2 THEN 70
80 FOR I=1 TO 14 :: CALL COLOR(I,1,1)::
NEXT I
90 ON BILD GOSUB 600,750,970,1190,1410,1
660,1910,2160,2410,2710
100 CALL COLOR(9,FA1,16):: CALL COLOR(10
,FA2,1,11,16,1,12,16,1)
110 FOR I=5 TO 8 :: CALL COLOR(I,16,5)::
 NEXT I :: CALL COLOR(3,16,14,4,16,14,2,
16,5)
120 CALL SPRITE(#1,129,10,X*8-7,Y*8-7)
130 DISPLAY AT(24,1):"SCORE "&STR$(SCORE
)&"   RUBBER BALLS "&STR$(MEN)
140 CALL KEY(0,K,S):: CALL LOCATE(#1,X*8
-7,Y*8-7)
150 IF K=83 THEN 220 ELSE IF K=68 THEN 2
50
160 IF K=14 THEN 20
170 CALL PATTERN(#1,129)
180 CALL GCHAR(X+1,Y,G):: IF G=99 THEN 2
80 ELSE IF G=98 THEN 310 ELSE IF G=32 TH
EN 350
190 IF G=100 THEN 570
200 CALL GCHAR(X-1,Y,PO):: IF PO<>32 THE
N 140 ELSE X=X-1
210 GOTO 140
220 Y=Y-1 :: CALL GCHAR(X,Y,GH):: IF GH=
97 OR GH=98 THEN Y=Y+1 ELSE IF GH=112 TH
EN 500 ELSE IF GH=104 THEN 560
230 CALL PATTERN(#1,132):: CALL GCHAR(X+
1,Y,T):: IF T=105 THEN 550 ELSE IF T=32
THEN 350
240 GOTO 140
250 Y=Y+1 :: CALL GCHAR(X,Y,GF):: IF GF=
97 THEN Y=Y-1 ELSE IF GF=112 THEN 500 EL
SE IF GF=104 THEN 560
260 CALL PATTERN(#1,131):: CALL GCHAR(X+
1,Y,U):: IF U=105 THEN 550 ELSE IF U=32
THEN 350
270 GOTO 140
280 FOR I=2000 TO 110 STEP -500 :: CALL
SOUND(-10,I,0):: NEXT I :: X=X2 :: Y=Y2
:: CALL LOCATE(#1,X*8-7,Y*8-7)
290 FOR I=0 TO 30 STEP 5 :: CALL SOUND(-
10,-1,I,110,I):: NEXT I
300 GOTO 140
310 CALL GCHAR(X+1,Y,H):: IF H=32 THEN 3
50 ELSE IF H=105 THEN 550 ELSE IF (H=97
OR H=104 OR H=100 OR H=99)THEN 140
320 CALL SOUND(-10,-1,0,300,0,110,0,115,
0)
330 X=X+1
340 CALL HCHAR(X,Y,32):: CALL LOCATE(#1,
X*8-7,Y*8-7):: GOTO 310
350 A=0
360 CALL GCHAR((X+1+A),Y,F):: IF F=97 OR
 F=98 OR F=99 OR F=104 OR F=100 THEN 390
 ELSE IF F=105 THEN 550
370 A=A+1 :: IF A>4 THEN 410
380 CALL LOCATE(#1,(A+X)*8-7,Y*8-7):: GO
TO 360
390 FOR I=0 TO 30 STEP 8 :: CALL SOUND(1
0,-1,I,210+X*10,I):: NEXT I
400 X=X+A :: GOTO 140
410 X=X+A :: CALL PATTERN(#1,133)
420 CALL GCHAR(X+1,Y,R):: IF R>=97 THEN
440
430 X=X+1 :: CALL SOUND(-10,1000+(X*100)
,0,-1,0,200,0,110,0):: CALL LOCATE(#1,X*
8-7,Y*8-7):: GOTO 420
440 CALL LOCATE(#1,X*8-7,Y*8-7):: FOR I=
0 TO 30 :: CALL SOUND(-10,110,I,300,I,40
0,I,-2,I):: NEXT I
450 MEN=MEN-1 :: IF MEN=0 THEN 2660
460 CALL KEY(0,K,S):: IF S=0 THEN 460
470 CALL DELSPRITE(ALL):: CALL SOUND(400
,110,0,-1,0,200,0)
480 OK=0
490 GOTO 90
500 FOR Z=1 TO 5 :: FOR I=2000 TO 110 ST
EP -700 :: CALL SOUND(-10,I,0,-1,0):: NE
XT I :: NEXT Z :: CALL HCHAR(X,Y,32)
510 FOR I=0 TO 30 :: CALL SOUND(10,110,I
,-1,I):: NEXT I
520 SCORE=SCORE+100+(BILD*10)
530 OK=OK+1 :: IF OK=KEYS THEN 60
540 GOTO 120
550 X=X+A :: CALL PATTERN(#1,133):: CALL
 LOCATE(#1,X*8-7,Y*8-7):: GOTO 440
560 CALL PATTERN(#1,129):: FOR I=0 TO 30
 STEP 10 :: CALL SOUND(10,-1,I,210+X*10,
I):: NEXT I :: X=X-1 :: GOTO 140
570 CALL SCREEN(12):: FOR I=0 TO 30 STEP
 7 :: CALL SOUND(100,-6,I):: NEXT I :: C
ALL DELSPRITE(#1)
580 CALL SCREEN(2)
590 GOTO 450
600 !B1
610 A$(1)="a                          aa"
 :: A$(2)="a                          aa"
620 A$(3)="abbbbbbbbbaaaaabaaaaaaabaaaa"
 :: A$(4)="abbbbbbbbbaaaaabaaaaaaabaaaa"
630 A$(5)="a          aa              aa"
 :: A$(6)="a          aa              aa"
640 A$(7)="ah         aa   h        h aa"
 :: A$(8)="a          aa              aa"
650 A$(9)="a          aa            h aa"
 :: A$(10)="a h        aa  h           aa
"
660 A$(11)="a         aa               aa
```

```
" :: A$(12)="a          aa                a
a"
670 A$(13)="a  h       aa h              aa
" :: A$(14)="a         paa               pa
a"
680 A$(15)="aaabaaaaaaaaaaaahhhhhhaaaaaa
" :: A$(16)="aaabaaaaaaaaaaa        aaaaa
a"
690 A$(17)="a                      h     aa
" :: A$(18)="a                            a
a"
700 A$(19)="a                     h      aa
" :: A$(20)="a                            pa
a"
710 A$(21)="aaaaaaaaaaaaaaaaaaacaaaaaaaa
" :: A$(22)="aaaaaaaaaaaaaaaaaaaaaaaaaa
a"
720 X=20 :: Y=4 :: X2=2 :: Y2=23 :: KEYS
=3
730 GOSUB 2650
740 RETURN
750 !B2
760 A$(1)="a                          aa  aa"
770 A$(2)="a                          a  paa"
780 A$(3)="aaaabbaaabbaaaabbbbbbaibaaa"
790 A$(4)="aaaabbaaabbaaaabbbbbbaabaaa"
800 A$(7)="aiiibiiibbbaaaah h"
810 A$(8)="aaaabaaabbbaaaa    i"
820 A$(9)="ap              a"
830 A$(10)="aaaaaaaa aaiiia"
840 A$(11)="a                h  i" :: A$(
12)="a"
850 A$(13)="a        h"
860 A$(14)="a         h"
870 A$(15)="a          h    h h"
880 A$(16)="a           h  h"
890 A$(17)="a"
900 A$(18)="acbbbbbaaaabbbbaa"
910 A$(19)="aabbbbbaaaabbbbaah"
920 A$(20)="a                    haaaaaaaii
"
930 A$(21)="aaiiiiiiiiiiiiiiiiiiiiiiaa
"
940 A$(22)="aaaaaaaaaaaaaaaaaaaaaaaaaa
"
950 X=2 :: Y=4 :: X2=2 :: Y2=27 :: KEYS=
2 :: GOSUB 2650
960 RETURN
970 !B3
980 FOR I=1 TO 4 :: A$(I)="a
              a" :: NEXT I
990 A$(5)="a                            pa"
1000 A$(6)="abh                       hbbbbba
"
1010 A$(7)="a  h                     h    bba
"
1020 A$(8)="ab aah                  h      a
"
1030 A$(9)="a      h              h     h  a
"
1040 A$(10)="aa      h            h     h
a"
1050 A$(11)="aah      h         h     h
a"
1060 A$(12)="a  h      h      h      h   b
a"
1070 A$(13)="a   h      h   h      h   a a
a"
1080 A$(14)="a    h    abbba    h    a a
a"
1090 A$(15)="a     h aaabbba   h    a b
a"
1100 A$(16)="ap                  h     a
a"
1110 A$(17)="abbbh  h         h      a
a"
1120 A$(18)="a    h  h      h       a
a"
1130 A$(19)="a     h  h    h        a
a"
1140 A$(20)="a      h     h         a
a"
1150 A$(21)="aciiaaaaaaaaaaaaaaaaaaaaca
a"
1160 A$(22)="aaaaaaaaaaaaaaaaaaaaaaaaaa
a"
1170 X=9 :: Y=4 :: X2=13 :: Y2=15 :: KEY
S=2 :: GOSUB 2650
1180 RETURN
1190 !B4
1200 FOR I=1 TO 3 :: A$(I)="a
                a" :: NEXT I
1210 A$(4)="abbbbbbbbbbbbbbbbbbbbbh   a
"
1220 A$(5)="abbbbbbbbbbbbbbbbbbbbb h  a
"
1230 A$(6)="a      iiiiii  ii      h   a
"
1240 A$(7)="a            bb         h  a
"
1250 A$(8)="a h    hhhh  iiiiiiiii h   a
"
1260 A$(9)="a  h       h p          h  a
"
1270 A$(10)="a           hbbbbbbbbb h
a"
1280 A$(11)="ab  bbbb                  h
a"
1290 A$(12)="a       b                 h
a"
1300 A$(13)="abah     bciiiiiiiiiiii h
a"
1310 A$(14)="a a h                     h
a"
1320 A$(15)="a   ph                   h
a"
1330 A$(16)="abbbbbbbbbbbbbbbbbbbbh
a"
1340 A$(17)="abbbbbbbbbbbbbbbbbbbb h
a"
1350 A$(18)="a iii  i  i  i  i      habb
a"
1360 A$(19)="a                       h a
a"
1370 A$(20)="a                      hap
a"
1380 A$(21)="aaaaaaaaaaaaaaaaaaaaaaaaaa
a" :: A$(22)="aaaaaaaaaaaaaaaaaaaaaaaaa
aa"
1390 X=20 :: Y=4 :: X2=9 :: Y2=21 :: KEY
S=3 :: GOSUB 2650
1400 RETURN
```

```
1410 !B5
1420 A$(1)="a                                  a
"
1430 A$(2)="a                                  a
"
1440 A$(3)="abbh              hbbbbhh     a
"
1450 A$(4)="abb h            h      p h    a
"
1460 A$(5)="a    hbbbbbbbbbbiiiiibbbbbba
"
1470 A$(6)="ah                                 a
"
1480 A$(7)="a hb                               a
"
1490 A$(8)="a                                  a
"
1500 A$(9)="a   babh                hbiibba
"
1510 A$(10)="a                     h
a"
1520 A$(11)="a    ii ii  p       h
a"
1530 A$(12)="a           bbbhhhhh      bb
a"
1540 A$(13)="a  b
a"
1550 A$(14)="a           iiibbbbbhh hh
a"
1560 A$(15)="a   iii          i    b    b
a"
1570 A$(16)="a                  h  i
a"
1580 A$(17)="a b              h
a"
1590 A$(18)="a b              h
a"
1600 A$(19)="a
a"
1610 A$(20)="aiii             bbbbhhhhhhh
a"
1620 A$(21)="aaaaa        iiiciiih
a"
1630 A$(22)="aaaaaaaaaaaaaaaaaaaaaaaaaaa
a"
1640 X=2 :: Y=4 :: X2=4 :: Y2=25 :: KEYS
=2 :: GOSUB 2650
1650 RETURN
1660 !B6
1670 A$(1)="aaaaaaaaaaaaaaaaaaaaaaaaaaa
"
1680 A$(2)="a           a                a
"
1690 A$(3)="a          pap               a
"
1700 A$(4)="a hddddddddabbbddddddaaaddh a
"
1710 A$(5)="a                            a
"
1720 A$(6)="aabddh                      ha
"
1730 A$(7)="a              hdh        h a
"
1740 A$(8)="a    aadddh        ddddddddda
"
1750 A$(9)="adddda        h ah          a
"
1760 A$(10)="aaaaaach   h   a ddh
a"
1770 A$(11)="a        h   aaaa    h
a"
1780 A$(12)="a            aaaad    ddddh
a"
1790 A$(13)="abbbbddddhdd   a
a"
1800 A$(14)="a              pa          h
a"
1810 A$(15)="a          hdddda d
a"
1820 A$(16)="a b d          a          h
a"
1830 A$(17)="a        hhhh  a
a"
1840 A$(18)="a              ad          h
a"
1850 A$(19)="a             h a
a"
1860 A$(20)="a  hbbbbbbddddda          h
a"
1870 A$(21)="aididddddddddda           p
a"
1880 A$(22)="aaaaaaaaaaaaaaaaaaaaaaaaaaa
a"
1890 X=10 :: Y=16 :: X2=10 :: Y2=19 :: K
EYS=4 :: GOSUB 2650
1900 RETURN
1910 !B7
1920 A$(1)="aaaaaaaaaaaaaaaaaaaaaaaaaaa
"
1930 A$(2)="a                            a
"
1940 A$(3)="a h         hdbabbbabb bbbbba
"
1950 A$(4)="a  ddb    h   a   a  h     a
"
1960 A$(5)="ah        ha   aiiia h i  iia
"
1970 A$(6)="a    h    a  ha   a         a
"
1980 A$(7)="a h     h a h a   aaaaa    a
"
1990 A$(8)="a     h   a   a       ah   a
"
2000 A$(9)="ah        ha   a      pahh pa
"
2010 A$(10)="a    h    acddddddddddddddd
a"
2020 A$(11)="a h     h
a"
2030 A$(12)="a     h bbbbbbbbbbbbbbb bbb
a"
2040 A$(13)="ah                      h
a"
2050 A$(14)="a    h                  h
a"
2060 A$(15)="a h                       h
a"
2070 A$(16)="a     h                 h
a"
2080 A$(17)="ah       hddddddddddddbbddd
a"
2090 A$(18)="a    h
a"
2100 A$(19)="a h     hi
```

```
a"
2110 A$(20)="a                          h
a"
2120 A$(21)="aaiiaaaaicaaiiddddddddddddd
a"
2130 A$(22)="aaaaaaaaaaaaaaaaaaaaaaaaaaa
a"
2140 X=20 :: Y=4 :: X2=9 :: Y2=18 :: KEY
S=2 :: GOSUB 2650
2150 RETURN
2160 !B8
2170 A$(1)="a                           a
"
2180 A$(2)="a                           a
"
2190 A$(3)="a habbbbbbbbbbbbbbbbbbbbbba
"
2200 A$(4)="a  abiiiiiiiiiiiiiiiiiiiiia
"
2210 A$(5)="ah ab                      pa
"
2220 A$(6)="a  abcddddddddddddddh hddda
"
2230 A$(7)="a haba                      a
"
2240 A$(8)="a  ab                   h   a
"
2250 A$(9)="ah aadddddddddbbabbbbbbb bba
"
2260 A$(10)="a  a             a       h
a"
2270 A$(11)="a habddddddddddaiii
a"
2280 A$(12)="a  abap                  h
a"
2290 A$(13)="ah ababbbbh            h
a"
2300 A$(14)="a  aba     bbbbh    hhh h
a"
2310 A$(15)="a haba            dddda  a
a"
2320 A$(16)="a  abaiiiiiiiiiihiii     a
a"
2330 A$(17)="ah abadddddddddddddddddddd
a"
2340 A$(18)="a  aba
a"
2350 A$(19)="a ha adabbbbbbbbbbbbbbbbbb
a"
2360 A$(20)="a  a a  iiiiiiiiiiiiiiiii
a"
2370 A$(21)="ah aiap
a"
2380 A$(22)="aaaaaaaaaaaaaaaaaaaaaaaaaa
a"
2390 X=21 :: Y=5 :: X2=18 :: Y2=9 :: KEY
S=3 :: GOSUB 2650
2400 RETURN
2410 !B9
2420 A$(1)="aaaaaaaaaaaaaaaaaaaaaaaaaa
"
2430 A$(2)="a              p  a   pa    a
"
2440 A$(3)="a hdddddddddd  a  aaaabdh a
"
2450 A$(4)="ah                a  h  aba ha
"
2460 A$(5)="a         aacabh abbhh aba  a
"
2470 A$(6)="a h     ha  ab   ab   abah a
"
2480 A$(7)="aaaaaaaaaa  ab h ab  haba  a
"
2490 A$(8)="a           ab   ab   aba ha
"
2500 A$(9)="ap           b  hab h aba  a
"
2510 A$(10)="aaaaaaaabbbbaa   ab   abah
a"
2520 A$(11)="a      a    aa h  b  haba
a"
2530 A$(12)="a  hh  a    aa   aa   aba h
a"
2540 A$(13)="a      a    aa  haa h aba
a"
2550 A$(14)="a i  h a h  aa   aa   abah
a"
2560 A$(15)="a      a    aa   aa  haba
a"
2570 A$(16)="a   h  a  h aa h aa   aba h
a"
2580 A$(17)="aiiiiiia         aa h  b
a"
2590 A$(18)="adddddddddddddiddddddddddah
a"
2600 A$(19)="aaaaaaaaaaaaaaaaaaaaaaaaaa
a"
2610 A$(20)="a                          h
a"
2620 A$(21)="aadddddddddddddddddddddddddd
a"
2630 A$(22)="aaaaaaaaaaaaaaaaaaaaaaaaaaa
a"
2640 X=20 :: Y=4 :: X2=11 :: Y2=6 :: KEY
S=3 :: GOSUB 2650 :: RETURN
2650 FOR I=1 TO 22 :: DISPLAY AT(I,1):A$
(I):: NEXT I :: FOR I=1 TO 22 :: A$(I)="
" :: NEXT I :: RETURN
2660 CALL CLEAR :: CALL DELSPRITE(ALL)::
 DISPLAY AT(12,6):"GAME OVER !"
2670 IF SCORE>HI THEN GOSUB 2790
2680 GOTO 2750
2690 DISPLAY AT(24,3):"PLAY AGAIN(Y/N)?"
2700 CALL KEY(0,K,S):: IF K=89 THEN 20 E
LSE IF K=78 THEN STOP ELSE 2700
2710 CALL CLEAR :: CALL DELSPRITE(ALL)::
 FOR I=2 TO 14 :: CALL COLOR(I,16,5):: N
EXT I
2720 FOR I=110 TO 600 STEP 100 :: CALL S
OUND(100,-1,0,I,0,110,0):: NEXT I
2730 DISPLAY AT(1,5):"CONGRATULATION !"
2740 DISPLAY AT(12,5):"YOU HAVE DONE IT"
2750 DISPLAY AT(16,1):"YOUR SCORE: "&STR
$(SCORE)
2760 IF SCORE>HI THEN GOSUB 2790
2770 DISPLAY AT(18,1):"HIGHSCORE: "&STR$
(HI)
2780 GOTO 2690
2790 CALL SOUND(1000,-1,0,110,0):: DISPL
AY AT(14,1):"YOU HAVE GOT THE HIGHSCORE"
 :: HI=SCORE :: RETURN
```

Ende des Listings

Hugo

Pyramiden färben, statt sterben!

Neuer Titel, altes Q-Bert! Unter diesem Slogan fiel uns Hugo in die Augen. Das reizvolle Spiel gibt es jetzt also auch auf dem TI-99/4A.

Mit einer kleinen Figur stehen Sie am höchsten Punkt der Pyramide. Ihre Aufgabe besteht nun in dem Umfärben aller Felder. Also Joystick anklemmen und den Feldern schön von oben nach unten einen neuen Anstrich verpassen. Leider ist das schwieriger, als es zu Beginn aussieht. Sie können nämlich nur runter laufen oder mit Druck auf den Feuerknopf waagrecht. Wenn Sie ein Feld vergessen haben, müssen Sie wieder mit dem Lift, rechts oder links, rauf fahren. Damit ist das Problem aber noch nicht gelöst, da alle Felder, die Sie nochmals überqueren, sich wieder umfärben. Hier ist die richtige Taktik erforderlich, sonst werden Sie nie fertig.

Doch damit nicht genug! Die Zeit läuft ebenfalls gegen Sie. Ganz zu schweigen von den schwarzen Viechern, die Ihnen nichts Gutes wollen. Lassen Sie sich nur nicht in Panik versetzen, sonst verlassen Sie möglicherweise ungewollt die Spielfläche und verlieren ein Leben..

Sie merken schon, daß Sie hier kein Kinderspiel erwartet. Viel Glück!

Das Listing

```
100 GOTO 270
110 CALL SPRITE(#1,128,7,8,110,#2,92,7,8
,126,#3,60,7,8,142):: DISPLAY AT(6,9):"P
RAESENTIERT:": : :TAB(13);"$%&'":TAB(13)
;")*+,"
120 DISPLAY AT(13,14):"ut":TAB(14);"dl":
TAB(13);"uemt":TAB(13);"dldl":TAB(12);"u
ememt":TAB(12);"dldldl":TAB(12);"!(!(!("
130 DISPLAY AT(22,1):" @ 1986 BY  JOERG
SENDROWSKI"
140 DISPLAY AT(24,2)BEEP:"DRUECKEN SIE D
IE FEUERTASTE"
150 CALL MELODY
160 CALL CLEAR  : CALL DELSPRITE(ALL)
170 DISPLAY AT(2,1):"HUGO. EIN REAKTIONS
SPIEL    FUER JUNG UND ALT. HUEPFEN  SIE
 MIT HUGO VON FELD ZU    FELD UND FAERBE
N SIE SO SO"
180 DISPLAY AT(6,1):"VIELE WIE MOEGLICH
UM. MIT  DEN FLUGSCHEIBEN KOENNEN SIEDAN
N WIEDER HOCHFLIEGEN."
190 DISPLAY AT(10,1):"MIT DEM JOYSTICK 2
 KOENNEN  SIE HUGO JEWEILS AUF DAS    NA
ECHST UNTERE FELD LENKEN."
200 DISPLAY AT(13,1):"WENN ER ABER AUF E
INER EBENEBLEIBEN SOLL- DANN MUESSEN  SI
E ZUSAETZLICH DEN FEUER   KNOPF DRUECKEN
. DIES MUSS"
210 DISPLAY AT(17,1):"MAN AUCH- WENN MAN
 AUF DIE  FLUGSCHEIBEN SPRINGEN WILL."
220 DISPLAY AT(20,1):"SPRINGEN SIE NICHT
 INS LEEREUND HALTEN SIE SICH VON DEN MO
NSTERN FERN."
230 DISPLAY AT(24,1):"DRUECKEN SIE DIE F
EUERTASTE"
240 CALL MELODY
250 CALL CLEAR
260 RETURN
270 CALL CLEAR :: CALL MAGNIFY(3):: RAND
OMIZE :: CALL SCREEN(2)
280 CALL CHAR(96,"0003070F193F3B1C0F0707
060E1E000000C0E0F098FCDC38F0E0E0607078")
290 A$="0103070F1F3F7FFF" :: B$="FFFEFCF
8F0E0C080" :: C$="FF7F3F1F0F070301" :: D
$="80C0E0F0F8FCFEFF"
300 CALL CHAR(100,D$&C$,108,A$&B$,116,D$
&A$,124,D$&C$,132,A$&B$,140,D$&A$,33,C$,
40,B$)
310 CALL CHAR(112,"03060C0C0D060F0C0D060
F0C0C060300C0603000C060C000C060C000C060C
")
320 CALL CHAR(136,"01070F1F3F3F3F1F0F070
1",138,"80E0F0F8FCFCFCF8F0E080")
330 CALL CHAR(104,"0000006010082E6B5F4C4
7404070000000000006081072D6FA32E202020E"
)
340 CALL CHARPAT(44,A$):: CALL CHAR(35,"
3CDBFFDBE73C2466",64,"3C4299A1A199423C",
45,A$)
350 CALL CH(72,36,41):: CALL CH(85,37,42
):: CALL CH(71,38,43):: CALL CH(79,39,44
)
360 CALL CHAR(128,"3048848040610A04844A3
1000004020100000609 0F884926000086492929 4
986")
370 CALL CHAR(92,"0000516A4A4A4A41182173
2121212120404 0C6494F4849E600008000000000
C0")
380 CALL CHARPAT(63,A$):: CALL CHAR(91,A
$)
390 CALL CHAR(60,"0000586040404040",61,"
",62,"",63,"")
400 CALL COLOR(1,5,1,2,8,1,9,5,3,10,8,3,
11,3,1,12,5,11,13,8,11,14,11,1)
410 CALL COLOR(3,5,1,4,5,1,5,14,1,6,14,1
,7,14,1,8,14,1)
420 GOSUB 110
430 DISPLAY AT(2,14):"ut":TAB(14);"dl":T
AB(13);"uemt":TAB(13);"dldl":TAB(12);"ue
memt":TAB(12);"dldldl":TAB(11);"uememem t
":TAB(11);"dldldldl"
440 DISPLAY AT(10,10):"uemememem t":TAB(1
0);"dldldldldl":TAB(9);"uemememememt":TA
B(9);"dldldldldldl":TAB(8);"uemememememe
mt"
450 DISPLAY AT(15,8):"dldldldldldldldl":TA
B(7);"uemememememem t":TAB(7);"dldldldl
dldldldl":TAB(6);"uemememememememt"
460 DISPLAY AT(19,6):"dldldldldldldldldl
":TAB(5);"uemememememememememt":TAB(5);"
dldldldldldldldldldldl":TAB(5);"!(!(!(!(!(
```

```
!(!(!(!(!("
470 CALL SPRITE(#10,136,6,155,33,#11,136
,6,155,209)
480 DISPLAY AT(2,2)SIZE(10):"SENDERSOFT"
 :: DISPLAY AT(5,3)SIZE(8):"$ % & '" ::
DISPLAY AT(6,3)SIZE(8):") * + ,"
490 DISPLAY AT(9,3)SIZE(6):"@ 1986"
500 DISPLAY AT(2,17):"SCORE:00000" :: DI
SPLAY AT(4,19):"# # # # #" :: DISPLAY AT
(6,21):"TIME:20"
510 TI=20
520 SB=4 :: SA,ZA=121 :: DIM F1(10,10),F
2(10,10):: FOR I=1 TO 10 :: FOR J=1 TO I
 :: F1(I,J)=SB :: F2(I,J)=ZA :: ZA=ZA+16
 :: NEXT J :: SB=SB+16 :: SA=SA-8
530 ZA=SA :: NEXT I
540 GOTO 550
550 M,B,A=1 :: CALL SPRITE(#1,96,15,4,12
1)
560 ON ERROR 940
570 DISPLAY AT(24,1):"DRUECKEN SIE DIE F
EUERTASTE" :: CALL SPRITE(#1,96,15,4,121
):: CALL MELODY :: DISPLAY AT(24,1)
580 FOR I=3 TO 1 STEP -1 :: CALL SOUND(1
00,110,I*3,550,I*6,1100,I):: NEXT I
590 FOR R=1 TO 4
600 CALL COINC(ALL,C):: IF C=-1 THEN 610
 ELSE 620
610 CALL COINC(#2,#3,10,C):: IF C=-1 THE
N 660 ELSE CALL TOD :: B,A,M=1 :: GOTO 5
90
620 CALL JOYST(2,X,Y):: CALL KEY(2,K,S):
: IF S=0 THEN IF X=-4 THEN B=B+1 ELSE IF
 X=4 THEN B=B+1 :: A=A+1 ELSE 850 ELSE 6
40
630 GOTO 650
640 IF X=-4 THEN A=A-1 ELSE IF X=4 THEN
A=A+1
650 ON M GOTO 660,670,690,740
660 CALL LOCATE(#1,F1(B,A),F2(B,A)):: CA
LL ST(B,A):: GOTO 770
670 Q=Q+1 :: IF Q=11 THEN M=1 :: CALL DE
LSPRITE(#2):: CALL LOCATE(#1,F1(B,A),F2(
B,A)):: CALL ST(B,A):: GOTO 770
680 W=W+INT(RND*2):: CALL LOCATE(#1,F1(B
,A),F2(B,A),#2,F1(Q,W),F2(Q,W)):: CALL S
T(B,A):: CALL SOUND(-1,110,15,990,8,-7,2
):: GOTO 770
690 Q=Q+1 :: U=U+1 :: IF Q=11 AND U<11 T
HEN M=4 :: CALL DELSPRITE(#2):: V=V+INT(
RND*2):: CALL LOCATE(#1,F1(B,A),F2(B,A),
#3,F1(U,V),F2(U,V)):: CALL ST(B,A):: GOT
O 770
700 IF U=11 AND Q<11 THEN M=2 :: CALL DE
LSPRITE(#3):: W=W+INT(RND*2):: CALL LOCA
TE(#1,F1(B,A),F2(B,A),#2,F1(Q,W),F2(Q,W)
):: CALL ST(B,A):: GOTO 770
710 IF U+Q=22 THEN M=1 :: CALL DELSPRITE
(#2,#3):: CALL LOCATE(#1,F1(B,A),F2(B,A)
):: CALL ST(B,A):: GOTO 770
720 W=W+INT(RND*2):: V=V+INT(RND*2):: CA
LL LOCATE(#1,F1(B,A),F2(B,A),#2,F1(Q,W),
F2(Q,W),#3,F1(U,V),F2(U,V)):: CALL ST(B,
A)
730 CALL SOUND(-1,110,14,550,5,1100,8,-6
,0):: GOTO 770
740 U=U+1 :: IF U=11 THEN M=1 :: CALL DE
LSPRITE(#3):: CALL LOCATE(#1,F1(B,A),F2(
B,A)):: CALL ST(B,A):: GOTO 770
750 V=V+INT(RND*2):: CALL LOCATE(#1,F1(B
,A),F2(B,A),#3,F1(U,V),F2(U,V)):: CALL S
T(B,A):: CALL SOUND(-1,330,15,880,8,-6,2
):: GOTO 770
760 V=V+INT(RND*2):: CALL LOCATE(#3,F1(U
,V),F2(U,V)):: CALL SOUND(-1,330,15,880,
8,-6,2):: GOTO 770
770 IF B=1 THEN M=1 :: GOTO 590
780 NEXT R
790 TI=TI-1 :: DISPLAY AT(6,26):TI :: IF
 TI=0 THEN 1070
800 IF M=1 THEN M=2 :: Q,W=1 :: CALL SPR
ITE(#2,112,2,4,121):: GOTO 590
810 IF M=2 THEN M=3 :: U,V=1 :: CALL SPR
ITE(#3,104,2,4,121):: GOTO 590
820 IF M=4 THEN M=3 :: Q,W=1 :: CALL SPR
ITE(#2,112,2,4,121):: GOTO 590
830 GOTO 590
840 CALL SOUND(-1,110,14,550,5,1100,8,-6
,0):: GOTO 770
850 ON M GOTO 770,860,880,920
860 Q=Q+1 :: IF Q=11 THEN M=1 :: CALL DE
LSPRITE(#2)ELSE W=W+INT(RND*2):: CALL LO
CATE(#2,F1(Q,W),F2(Q,W)):: CALL SOUND(-1
,110,15,990,8,-7,2)
870 GOTO 770
880 Q=Q+1 :: U=U+1 :: IF Q=11 AND U<11 T
HEN M=4 :: CALL DELSPRITE(#2):: V=V+INT(
RND*2):: CALL LOCATE(#3,F1(U,V),F2(U,V))
:: GOTO 770
890 IF U=11 AND Q<11 THEN M=2 :: CALL DE
LSPRITE(#3):: W=W+INT(RND*2):: CALL LOCA
TE(#2,F1(Q,W),F2(Q,W)):: GOTO 770
900 IF U+Q=22 THEN M=1 :: CALL DELSPRITE
(#2,#3):: GOTO 770
910 W=W+INT(RND*2):: V=V+INT(RND*2):: CA
LL LOCATE(#2,F1(Q,W),F2(Q,W),#3,F1(U,V),
F2(U,V)):: CALL SOUND(-1,110,13,550,6,11
00,8,-6,0):: GOTO 770
920 U=U+1 :: IF U=11 THEN M=1 :: CALL DE
LSPRITE(#3)ELSE V=V+INT(RND*2):: CALL LO
CATE(#3,F1(U,V),F2(U,V)):: CALL SOUND(-1
,330,15,770,8,-6,2)
930 GOTO 770
940 IF B=10 THEN 980 ELSE CALL TOD :: B,
A,M=1
950 CALL LOCATE(#1,F1(B,A),F2(B,A))
960 ON ERROR 940
970 GOTO 590
980 IF A=0 THEN 990 ELSE 1030
990 CALL LOCATE(#1,147,33)
1000 CALL MOTION(#1,-8,5,#10,-8,5):: FOR
 I=24.5 TO 1 STEP -.5 :: CALL SOUND(-100
,110,I,550+I*100,I,1100*I,I):: NEXT I
1010 CALL MOTION(#1,0,0,#10,0,0):: CALL
LOCATE(#10,155,33)
1020 FOR I=440 TO 990 STEP 110 :: CALL S
OUND(50,I,1):: CALL SC(10):: NEXT I :: B
,A=1 :: GOTO 960
```

```
1030 CALL LOCATE(#1,147,209)
1040 CALL MOTION(#1,-8,-5,#11,-8,-5):: F
OR I=24.5 TO 1 STEP -.5 :: CALL SOUND(-1
00,110,I,550+I*100,I,1100*I,I):: NEXT I
1050 CALL MOTION(#1,0,0,#11,0,0):: CALL
LOCATE(#11,155,209)
1060 GOTO 1020
1070 FOR I=550 TO 990 STEP 110 :: CALL S
OUND(10,I,1):: NEXT I :: FOR I=2 TO 10 :
: FOR J=1 TO I
1080 X=13-I+2*J :: Y=2*I :: CALL GCHAR(Y
,X+2,C):: IF C>120 THEN CALL ST(I,J):: C
ALL SOUND(100,550,5,990,9,1100,5):: CALL
 SC(10)
1090 NEXT J :: NEXT I
1100 TI=20 :: CALL DELSPRITE(#1,#2,#3)::
 B,A,M=1
1110 FOR S=1 TO 2 :: X=INT(RND*8)+3 :: Y
=INT(RND*X)+1
1120 Y=(15-X)+2*Y :: X=X*2 :: FOR I=X TO
 X+2 :: FOR J=Y TO Y+1 :: CALL GCHAR(I,J
,C)
1130 IF C=140 OR C=141 OR C=108 OR C=100
 OR C=124 OR C=132 THEN CALL VCHAR(I,J,3
2)
1140 IF I>X AND C=33 OR I>X AND C=40 OR
I=X AND C=116 OR I=X AND C=117 THEN CALL
 VCHAR(I,J,32)
1150 IF I=X THEN PA=40 :: AP=33 ELSE PA=
117 :: AP=116
1160 IF C=109 OR C=133 THEN CALL VCHAR(I
,J,PA)ELSE IF C=101 OR C=125 THEN CALL V
CHAR(I,J,AP)
1170 NEXT J :: NEXT I
1180 FOR I=110 TO 1100 STEP 110 :: CALL
SOUND(10,I,1):: NEXT I
1190 NEXT S
1200 FAR=FAR+1 :: IF FAR=4 THEN FAR=1
1210 ON FAR GOTO 1230,1240,1220
1220 CALL COLOR(12,5,11,13,8,11,14,11,1)
:: GOTO 570
1230 CALL COLOR(12,5,7,13,8,7,14,7,1)::
GOTO 570
1240 CALL COLOR(12,5,9,13,8,9,14,9,1)::
GOTO 570
1250 SUB ST(X,Y):: S=13-X+2*Y :: Z=X*2 :
: CALL GCHAR(Z,S+2,C1):: CALL GCHAR(Z,S+
3,C2)
1260 IF C1=40 OR C1=32 THEN CALL TOD ::
X,Y=1 :: SUBEXIT
1270 Z1$=CHR$(132):: Z2$=CHR$(133):: Z3$
=CHR$(140):: Z4$=CHR$(141)
1280 IF C1=117 AND C2=116 THEN A$=Z4$&Z3
$ :: GOTO 1330
1290 IF C1=117 THEN A$=Z4$&"}" :: B$="|"
&Z1$ :: GOTO 1330 ELSE IF C2=116 THEN A$
=Z2$&Z3$ :: B$="|"&Z1$ :: GOTO 1330
1300 IF C1=109 THEN A$=Z2$&"}" :: B$="|"
&Z1$ :: GOTO 1330 ELSE IF C1=141 THEN A$
="ue" :: B$="dl" :: GOTO 1330
1310 IF C1=141 AND C2=140 THEN A$="ut" :
: GOTO 1330
1320 IF C2=140 THEN A$="mt" :: B$="dl" :
: GOTO 1330 ELSE IF C1=133 THEN A$="me"
:: B$="dl" :: GOTO 1330
1330 DISPLAY AT(Z,S)SIZE(2)BEEP:A$ :: DI
SPLAY AT(Z+1,S)SIZE(2):B$ :: SUBEND
1340 SUB TOD
1350 IF M=0 THEN M=5
1360 M=M-1 :: IF M=0 THEN CALL OVER
1370 CALL POSITION(#1,B,A):: IF A<121 TH
EN CALL MOTION(#1,0,-10):: Z=-5 ELSE CAL
L MOTION(#1,0,10):: Z=5
1380 FOR I=1 TO 80 :: NEXT I
1390 CALL MOTION(#1,10,Z):: FOR I=1 TO 1
5 STEP .5 :: CALL SOUND(-100,2000-I*100,
I,880,I,550-I*10,I)
1400 NEXT I :: CALL DELSPRITE(#1,#2,#3):
: DISPLAY AT(4,19):RPT$("# ",M)
1410 RESTORE 1440
1420 FOR K=1 TO 11 :: READ A,B :: FOR I=
0 TO B STEP 2 :: CALL SOUND(-500,A,I,A,I
,A,I):: NEXT I
1430 NEXT K
1440 DATA 131,10,131,8,131,2,131,6,165,8
,147,2,147,4,131,2,131,4,123,2,131,10
1450 CALL SPRITE(#1,96,15,4,121):: SUBEN
D
1460 SUB SC(X):: S=S+10 :: DISPLAY AT(2,
23):RPT$("0",5-LEN(STR$(S)))&STR$(S):: S
UBEND
1470 SUB OVER
1480 DISPLAY AT(12,8):"G A M E  O V E R"
 :: DISPLAY AT(11,8):: DISPLAY AT(13,8)
1490 DISPLAY AT(24,1):"DRUECKEN SIE DIE
FEUERTASTE"
1500 CALL KEY(2,K,S):: IF S=0 THEN 1500
ELSE CALL CLEAR :: CALL DELSPRITE(ALL)::
 RUN 400
1510 SUBEND
1520 SUB CH(X,Y,Z)
1530 CALL CHARPAT(X,A$):: B$="" :: FOR I
=1 TO 15 STEP 2 :: B$=B$&RPT$(SEG$(A$,I,
2),2):: NEXT I
1540 CALL CHAR(Y,SEG$(B$,1,16),Z,SEG$(B$
,17,16)):: SUBEND
1550 SUB MELODY
1560 RESTORE 1590
1570 READ A,B :: IF A=0 THEN 1560 ELSE I
F A=1 THEN CALL SOUND(-1,20000,30)ELSE C
ALL SOUND(-800,A,3,B,6)
1580 CALL KEY(2,K,S):: IF S=0 THEN 1570
ELSE CALL SOUND(-10,20000,30,20000,30)::
 SUBEXIT
1590 DATA 659,262,659,262,784,392,1,1,78
4,330,784,330,784,330
1600 DATA 1,1,784,330,698,175,698,175,88
0,262,1,1,880,175,880,175
1610 DATA 1,1,880,262,988,196,988,196,1,
1,988,294,880,196,880,196,988,196,988,19
6
1620 DATA 1047,262,1047,262,1175,196,117
5,196,1319,262,1319,262,1319,262,1319,26
2
1630 DATA 1,1,1,1,0,0
1640 SUBEND
```

Ende des Listings

```
81,20,K7)
(NI) 1760 IF PEEK(53279)=K3 THEN P(K1)=P
(K1)+K1:POSITION K3,K5:? PL$(P(K1)*K10+K
1,P(K1)*K10+K10)
(MM) 1770 IF PEEK(53279)=K5 THEN P(K2)=P
(K2)+K1:POSITION 29,K5:? PL$(P(K2)*K10+K
1,P(K2)*K10+K10)
(DL) 1780 IF P(K1)>K2 THEN P(K1)=K0
(DO) 1790 IF P(K2)>K2 THEN P(K2)=K0
(JA) 1800 IF PEEK(53279)=K6 THEN 1820
(JB) 1810 NEXT I:GOTO 1740
(HF) 1820 POKE 706,14:POKE 707,K0:Q=USR(
MOVE,K2,PMB,SPIN,66,20,K7):Q=USR(MOVE,K3
,PMB,SPIN,181,20,K7)
(PL) 1830 POSITION 9,K2:? "
        ":POSITION K3,K6:? "          ":PO
SITION 29,K6:? "          "
(GP) 1840 GOTO 90
(LC) 1850 REM
(MO) 1860 Q=USR(MOVE,K0,PMB,K0,K0,K0,K0)
:FOR I=K1 TO K8:Z=PEEK(711):POKE 711,Z+(
Z<K8)-(Z>K8):NEXT I
(IA) 1870 IF SC(K1)=SC(K2) THEN POSITION
 14,21:? "IT IS A TIE.":GOTO 1910
(EE) 1880 POSITION 12,21:? "SPIELER";
(OE) 1890 IF SC(K1)>SC(K2) THEN ? "EINS
SIEGT!"
(OM) 1900 IF SC(K2)>SC(K1) THEN ? "ZWEI
SIEGT!"
(KN) 1910 FOR I=K0 TO 500:NEXT I
(CH) 1920 FOR I=704 TO 711:FOR J=PEEK(I)
 TO K0 STEP -K1:POKE I,J:NEXT J:NEXT I
(OI) 1930 ? CHR$(125):POKE 559,K0:GOTO 1
470
```

Ende des Listings

Jumping Miner Leider springt auch der Zeiger!

Insgesamt müssen Sie fünf Minen bewältigen. Da kommt einem die Übungsmine zu Beginn doch sehr gelegen. Sammeln Sie hier genügend Erfahrungen, denn schon in der zweiten Mine kommt es auf eine sehr genaue Sprungtechnik an. Im dritten Level ist Ihr ganzes Können auf der Rutsche gefragt. Erinnerungen aus der Kindheit werden wachgerüttelt; nutzen Sie Ihre Erfahrungen. In der vierten Mine sind wir dann schon bei den Tücken angelangt, wo es kein Zurück mehr gibt. Die letzte Station ist wirklich die absolute Krönung. Hier schlakkert selbst der Computer-Freak mit den Ohren.

Der Jumping-Miner kann noch zusätzlich Punkte sammeln, indem er durch einen gezielten Sprung genau auf einem Erdhügel landet. Zur Belohnung wächst aus dem Erdhaufen eine Pflanze. Weitere Extra-Punkte können durch Abhängen der Lampen kassiert werden. Lediglich unter die Lampe stellen und nach oben springen, dann rattern die Punkte auf der Anzeige.

```
100 ! ***********************      *  JUM
PING - MINER   *      *    (c) 1985
   *      *   W.H.DOELTSCH     *      **
********************
110 ! *                    *      * 6108
 WEITERSTADT 3 *      *
   *      * A.D.HINTERSTEIN 10 *      *
                 *
120 ! ***********************      * BENO
ETIGTE GERAETE:*      *     TI-99/4A
   *      *      X-BASIC       *      *
    JOYSTICK 1     *
125 ! ***********************      * 1587
 BZW 503 BYTES *      *          FREE
   *      * HOECHSTE VORSICHT  *
126 ! *BEI DEN DATA-ZEILEN!*      ******
****************
130 CALL CLEAR :: GOTO 670
140 CALL KEY :: CALL SOUND :: CALL GCHAR
 :: CALL SPRITE :: CALL COLOR :: CALL HC
HAR :: CALL VCHAR :: CALL DELSPRITE
150 CALL MOTION :: CALL CHAR :: CALL LOC
ATE :: CALL PATTERN :: CALL JOYST :: M1,
M2,M3,R,X,Y,Z,S,I,II,A,B,TA,ST,SC,NSC,TI
160 LEV,LIFE,TOTAL,FLAG,A$,B$,C$,D$,E$
170 DATA 0000003E3E7E3E1C,1C3E7E7E7E3C00
00,040408100000040C,001C7E0610041800
180 DATA 2020100800002030,00387E60082018
00,0000007C7C7E7C38,387C7E7E7E3C0000,D0F
365971A101038,,,
190 DATA 004042020000740E,00183C7E000000
00,0000003C3C7E3C3C,3C7E7E7E7E3C0000
200 DATA 04041C0000001236,20203800000048
6C,,0002424000002E70
210 DATA 04087000000043C1,20100E000000C2
83
220 DATA 33,00FFFF0000FFFF00,88,0103070F
1F3F7FFF,89,80C0E0F0F8FCFEFF
230 DATA 91,FFFEFCF8F0E0C080,92,FF7F3F1F
0F070301,35,40C0C04040C0C040,36,02030302
02030302,37,0000000000003CFF
240 DATA 118,103838387CFE7C00
250 DATA 370,392,440,370,294,370,440,370
,294,370,2,440,2,587,2,587
260 DATA 554,494,440,370,294,370,440,370
,294,370,2,392,2,330,2,330
270 DATA 370,392,440,370,294,370,440,370
,294,370,2,440,2,587,2,587
280 DATA 659,784,740,880,740,587,659,784
,659,554,2,587,2,587,2,587,99
290 DATA 294,440,370,440,294,440,370,440
,587,440,740,440,659,440,740,440
300 DATA 784,440,740,440,659,587,554,587
,659,587,554,494,440,392,370,330
310 DATA 294,440,370,440,294,440,370,440
,587,440,740,440,659,440,740,440
320 DATA 784,440,740,440,659,587.554,494
,554,587,659,784,740,587,2,587,99
330 DATA "((((((((((((((((((((((((((((*
$!#v  $!# v   $!#      v Z  $!#   $!#
```

```
  $!#          Z  $!#%  $!# %   $!#    %
 Z"
340 DATA "(((((( ((((( (((((,((((((,!#
                  XZ[    XZ!#
   XZ[    XZ[!#   v        v   XZ[     .
( "
350 DATA "!#                XZ[       \ZY!#
 %        %   XZ[%      % \Z((((((    ((.(
(((((  ((((((* v $!#       \ZY  v    $!#
 Z"
360 DATA "   $!#        \ZY       $!#  Z
 $!# %       \ZY  %  $!# %Z (((((((((  ((
((((((((((((,   v      v   $!#     v
XZ"
370 DATA "              $!#          XZ[.
%     %    $!# %    % XZ[ ((((((((((((((
((((((((((((((("
380 DATA 5,2,40,5,31,40,6,31,90,7,31,90,
8,31,90,9,2,40,9,31,40,10,2,36,10,31,91,
11,2,36,12,2,36
390 DATA 13,2,36,14,2,36,14,31,89,15,2,4
0,15,31,40,16,31,90,17,31,90,18,31,90,19
,31,40,20,31,91
400 DATA 23,2,40,23,31,40,138,3500
410 DATA "((((((        ((((((((  (((,($!
#  v         v  $!#     XZ[$!#     (   (
   $!#    XZ[ $!#    ( (    %  $!#   XZ[
  "
420 DATA "$!#  ((    ,(  (  $!#  ((,(  $!
#  v   XZ[    ( $!#  vXZ[  $!#     XZ[
   $!#  XZ[   $!#  % XZ[    %  $!# XZ[
% "
430 DATA "$!#  (.(((((((((,((((*((((((($!
#   \ZY  v   XZ[ v ZZ      $!#    \ZY
XZ[    ZZ     $!#  (  \ZY% XZ[     ZZ (
 ("
440 DATA "$!#    ((.((((((  (( ZZ   ( $!
#     XZ[         v ZZ  (  $!#    XZ[ v %
  %    ZZ    ($!#   XZ[    (  (    ZZ
( "
450 DATA "$!#  XZ[  (         ( ZZ  (  $!
# XZ[ %  (  %   (  ZZ (  %((((((((((((((
((((((((((((((("
460 DATA 5,2,40,5,31,40,13,31,40,18,31,4
0,23,2,40,23,31,40,116,4100
470 DATA "((.((((  (((,(((  ((((,(((((
\ZY      XZ[        XZ[v$!#   \ZY    XZ[
      XZ[  $!#   .((   ((,(       XZ[ % $
!#"
480 DATA "   \ZY    XZ[      XZ[  (((((
  \ZY %XZ[      XZ[   $!#       ,(((((,(
  XZ[    $!#     XZ[   XZ[   XZ[  %  $!#
 %"
490 DATA "% XZ[  %XZ[   XZ[  ((  (((((((
(((( ((((  XZ[       v $!#            XZ
[      (   $!#   %       XZ[    (     %$
!#"
500 DATA "   ( (((  XZ[ (        ((((( %
       XZ[      %    $!#   ( ((   XZ[
  (((((  $!# %       XZ[ (( (    v   (((
((("
510 DATA " ((   XZ[                $!#
  %XZ[    %        %   $!#((((((((((((((
((((((((((((((("
520 DATA 5,2,40,5,31,40,9,31,40,14,2,40,
13,31,40,17,31,40,20,2,40,20,31,40,23,2,
40,23,31,40,129,4700
530 DATA "(( (*((((((((((((*( (((,(((!#
  ZZ      $!#   ZZ   XZ[$!!#% ZZ  %   $!
#   ZZ  XZ[v$!((((,(( (   (((   ZZ XZ[
$!"
540 DATA "   XZ[v v   $!#   ZZXZ[  %$!v
XZ[       $!# % (((((( ((( XZ[  % %   ((
( (         v XZ[   ( (   $!#
  "
550 DATA "((( (      ( $!# ( (( (    %  v
          (((     %  (  (       %  (  $!
#     (        %  % (     $!#   (     ((
(,"
560 DATA " (  (   %   ((( %   (     XZ
  v   (   $!# (     % (XZ[  (       (  $!
#       (  *(   v %  %    ((( % % (    Z
Z "
570 DATA "    (  ( (  $!# ( (      ZZ  %
    %    %$!#      %   ZZ ((((((((((((((
((((((((((((((("
580 DATA 5,2,40,5,31,40,6,2,36,6,31,35,7
,2,36,7,31,35,8,2,40,8,31,35,9,31,35,10,
31,40
590 DATA 13,2,40,16,31,40,17,31,91,19,2,
40,23,2,40,23,31,40,134,4800
600 DATA "       (*((  ((((((.(((
   ( ZZv         \ZY  (         (  ZZ
 %   \ZY   (     (   ZZ  % (((  ( \ZY
  "
610 DATA "  (    (((((         \ZY v   (
   v          (  ((  \ZY   (       (  (
 v       \ZY   (((     v          %   (*(
  "
620 DATA "    (((    (   %   (    ZZ (
 v$!#    v(  (       (ZZv  (  $!#  (
     (    ZZ      $!#   %      %   v  ZZ
  "
630 DATA " (.(((( %*((((((*(       (((,
\ZY   (ZZ $!# ZZ   % (  XZ ( \ZY(  ZZ $!
# ZZ ( (   XZ[   (\ZY  ZZ $!# ZZ     (XZ
[ "
640 DATA " (   \ZY ZZ $!# ZZ ((  XZ[
 %  \ZYZZ $!# ZZ    XZ[ % ((((((((((((
((((((((((((("
650 DATA 7,31,40,12,2,40,17,31,40,18,2,4
0,18,31,91,22,2,40,23,2,40,23,31,40,119,
4900
660 !@P-
670 FOR I=96 TO 117 :: READ A$ :: CALL C
HAR(I,A$):: NEXT I :: FOR I=1 TO 9 :: RE
AD Z,A$ :: CALL CHAR(Z,A$):: NEXT I
680 A$="FFFF0FF00FF00FF0" :: B$="FFFFFFF
FFFFFFFFF"
690 CALL CHAR(40,A$,41,B$,42,A$,43,B$,44
,A$,45,B$,46,A$,47,B$,90,B$)
700 CALL COLOR(1,5,1,3,16,1,4,16,1,5,16,
1,6,16,1,7,16,1,8,14,2,2,11,2,10,9,1,11,
15,2)
710 LIFE=3 :: LEV=1 :: SC=0 :: GOTO 770
720 RESTORE 330 :: II=23 :: RETURN
730 RESTORE 410 :: II=6 :: RETURN
740 RESTORE 470 :: II=10 :: RETURN
```

```
750 RESTORE 530 :: II=16 :: RETURN
760 RESTORE 600 :: II=8 :: RETURN
770 GOSUB 1390 :: FLAG=0 :: ON LEV GOSUB
 720,730,740,750,760 :: DISPLAY AT(1,1)S
IZE(15):"SCORE: ";SC :: DISPLAY AT(1,18)
SIZE(10):"TIME: 0000"
780 READ A$,B$,C$,D$,E$ :: DISPLAY AT(5,
1):A$&B$:C$&D$:E$ :: FOR I=1 TO II :: RE
AD X,Y,Z :: CALL HCHAR(X,Y,Z):: NEXT I :
: READ TOTAL :: READ TI
790 Z=161 :: S=128 :: CALL DELSPRITE(ALL
):: M1,M2,M3=0
800 CALL SPRITE(#1,98,16,Z+8,S,#2,99,5,Z
,S,#3,96,10,Z,S,#4,97,13,Z+8,S)
810 CALL KEY(1,TA,ST):: IF TA=18 THEN GO
SUB 1350
820 TI=TI-2 :: DISPLAY AT(1,23):TI :: IF
 TI<=0 THEN GOTO 1090
830 CALL JOYST(1,X,Y):: R=((X+3*Y)/4)+5
:: ON R GOTO 830,870,830,840,810,850,830
,860,830
840 M1=98 :: IF M3=M1 THEN GOTO 890 ELSE
 CALL PATTERN(#1,M1,#2,99,#3,96,#4,97)::
 M2=112 :: GOTO 890
850 M1=100 :: IF M3=M1 THEN GOTO 890 ELS
E CALL PATTERN(#1,M1,#2,101,#3,102,#4,10
3):: M2=113 :: GOTO 890
860 CALL GCHAR(Z/8+2,S/8+1,I):: IF I<>33
 AND I<>40 AND I<>41 THEN GOTO 830 ELSE
GOTO 880
870 CALL GCHAR(Z/8+4,S/8+1,I):: IF I<>33
 AND I<>40 AND I<>41 THEN GOTO 830
880 M1=108 :: IF M3=M1 THEN GOTO 890 ELS
E CALL PATTERN(#1,M1,#2,109,#3,110,#4,11
1):: M2=115 :: GOTO 890
890 Y=-Y :: CALL KEY(1,TA,ST):: IF TA<>1
8 THEN 980
900 IF R=4 THEN M1=116 :: CALL PATTERN(#
1,M1)ELSE M1=117 :: CALL PATTERN(#1,M1)
910 CALL SOUND(60,-6,5):: FOR B=1 TO 3 :
: A=X+S :: IF (A<8)+(A>240)THEN M3=M2 ::
 CALL PATTERN(#1,M2):: GOTO 1110
920 S=X+S :: Z=Z-4 :: CALL LOCATE(#1,Z+8
,S,#2,Z,S,#3,Z,S,#4,Z+8,S):: TI=TI-2 ::
IF B=2 THEN GOTO 1140
930 NEXT B
940 FOR B=1 TO 7 :: A=X+S :: IF (A<8)+(A
>240)THEN M3=M2 :: CALL PATTERN(#1,M2)::
 GOTO 1110
950 Z=Z+4 :: S=X+S :: CALL LOCATE(#1,Z+8
,S,#2,Z,S,#3,Z,S,#4,Z+8,S):: TI=TI-2 ::
ON B GOTO 1170,960,1170,960,1170,960,117
0
960 NEXT B :: CALL PATTERN(#1,M2)
970 GOTO 1060
980 S=X+S :: Z=Y+Z :: IF S<8 OR S>240 TH
EN GOTO 990 ELSE 1000
990 IF S<8 THEN S=8 :: GOTO 830 ELSE IF
S>240 THEN S=240 :: GOTO 830
1000 CALL LOCATE(#1,Z+8,S,#2,Z,S,#3,Z,S,
#4,Z+8,S):: CALL PATTERN(#1,M2)
1010 CALL SOUND(1,-6,8):: TI=TI-10 :: DI
SPLAY AT(1,23):TI :: IF TI<=0 THEN GOTO
1090
1020 S=X+S :: Z=Y+Z :: CALL LOCATE(#1,Z+
8,S,#2,Z,S,#3,Z,S,#4,Z+8,S):: CALL SOUND
(1,-7,12):: CALL PATTERN(#1,M1):: M3=M1
:: GOTO 1110
1030 FOR I=1 TO 4 :: Z=Z+8 :: CALL LOCAT
E(#1,Z+8,S,#2,Z,S,#3,Z,S,#4,Z+8,S):: CAL
L GCHAR(Z/8+3,S/8+1,A)
1040 IF A>47 THEN GOTO 1050 ELSE ON A-31
 GOTO 1050,1050,1050,1050,1050,1050,1050
,1050,1130,830,1190,1190,1210,1210,1270,
1270
1050 NEXT I
1060 FOR I=1 TO 20 :: Z=Z+8 :: CALL LOCA
TE(#1,Z+8,S,#2,Z,S,#3,Z,S,#4,Z+8,S):: CA
LL GCHAR(Z/8+3,S/8+1,A)
1070 IF A=40 OR A=41 THEN GOTO 1090
1080 NEXT I
1090 DISPLAY AT(1,23):" 0000" :: CALL MO
TION(#1,-3,1,#2,-3,-1,#3,-2,2,#4,-2,-2):
: CALL SOUND(3000,-7,15):: FOR X=1 TO 50
0 :: NEXT X :: CALL DELSPRITE(ALL)
1100 LIFE=LIFE-1 :: TOTAL,TI=0 :: IF LIF
E=0 THEN GOTO 1480 ELSE GOTO 770
1110 CALL GCHAR(Z/8+3,S/8+1,I)
1120 IF I>47 THEN GOTO 1030 ELSE ON I-31
 GOTO 1030,830,830,1030,1030,830,830,830
,1130,830,1190,1190,1220,1220,1270,1270
1130 CALL HCHAR(Z/8+3,S/8+1,41):: SC=SC+
5 :: GOSUB 1340 :: GOTO 830
1140 CALL GCHAR(Z/8+2,S/8+1,R):: CALL GC
HAR(Z/8+3,S/8+1,I):: IF R=37 THEN GOTO 1
160 ELSE M3=M1 :: CALL PATTERN(#1,M2)
1150 IF I>47 THEN GOTO 930 ELSE ON I-31
GOTO 930,930,930,930,930,930,930,930,113
0,830,1190,1190,1220,1220,1270,1270
1160 M3=M1 :: CALL PATTERN(#1,M2):: CALL
 SOUND(100,880,5,1108,7,1318,9):: SC=SC+
100 :: DISPLAY AT(1,8)SIZE(10):SC :: CAL
L HCHAR(Z/8+2,S/8+1,104):: GOTO 1110
1170 CALL GCHAR(Z/8+2,S/8+1,R):: CALL GC
HAR(Z/8+3,S/8+1,I):: IF R=37 THEN GOTO 1
160 ELSE M3=M1 :: CALL PATTERN(#1,M2)
1180 IF I>47 THEN GOTO 960 ELSE ON I-31
GOTO 960,960,960,960,960,960,960,960,113
0,830,1190,1190,1220,1220,1270,1270
1190 CALL PATTERN(#1,108,#2,109,#3,110,#
4,111):: IF I=42 THEN CALL HCHAR(Z/8+3,S
/8+1,43):: GOSUB 1340
1200 Z=Z+8 :: CALL LOCATE(#1,Z+8,S,#2,Z,
S,#3,Z,S,#4,Z+8,S):: CALL SOUND(-200,100
0+Z,1)
1210 CALL GCHAR(Z/8+3,S/8+1,I):: IF I=90
 THEN GOTO 1200 ELSE M3=0 :: GOTO 1120
1220 CALL PATTERN(#1,116,#2,99,#3,96,#4,
97):: IF I=44 THEN CALL HCHAR(Z/8+3,S/8+
1,45):: GOSUB 1340
1230 Z=Z+8 :: CALL LOCATE(#1,Z+8,S,#2,Z,
S,#3,Z,S,#4,Z+8,S):: CALL SOUND(-200,100
0+Z,1)
1240 CALL GCHAR(Z/8+3,S/8+1,I):: IF I=90
 THEN S=S-8 :: GOTO 1230
1250 CALL PATTERN(#1,98):: IF FLAG=1 THE
N M3=0 :: GOTO 1110 ELSE CALL GCHAR(Z/8+
3,S/8,R):: IF R<>32 THEN M3=0 :: GOTO 11
```

```
10
1260 CALL HCHAR(Z/8+3,S/8+2,45):: FLAG=1
 :: GOSUB 1340 :: GOTO 1120
1270 CALL PATTERN(#1,117,#2,101,#3,102,#
4,103):: IF I=46 THEN CALL HCHAR(Z/8+3,S
/8+1,47):: GOSUB 1340
1280 Z=Z+8 :: CALL LOCATE(#1,Z+8,S,#2,Z,
S,#3,Z,S,#4,Z,S):: CALL SOUND(-200,1000+
Z,1)
1290 CALL GCHAR(Z/8+3,S/8+1,I):: IF I=92
 THEN S=S+8 :: GOTO 1280 ELSE CALL PATTE
RN(#1,100):: M3=0 :: GOTO 1120
1300 SC=SC+TI :: DISPLAY AT(1,8)SIZE(10)
:SC :: TI=0 :: DISPLAY AT(1,18)SIZE(10):
"TIME: 0000" :: RESTORE 290
1310 READ A :: IF A=99 THEN GOTO 1330 ÉL
SE IF A<110 THEN READ I ELSE I=A :: A=1
1320 CALL SOUND(INT(150*A),I,1,I+4,1,I+8
,1):: GOTO 1310
1330 CALL DELSPRITE(ALL):: LEV=LEV+1 ::
IF LEV>5 THEN GOTO 1480 ELSE 770
1340 DISPLAY AT(1,8)SIZE(10):SC :: TOTAL
=TOTAL-1 :: IF TOTAL=0 THEN GOTO 1300 EL
SE RETURN
1350 FOR I=1 TO 2 :: Z=Z-4 :: CALL LOCAT
E(#1,Z+8,S,#2,Z,S,#3,Z,S,#4,Z+8,S):: NEX
T I
1360 CALL GCHAR(Z/8+1,S/8+1,II):: IF II<
>118 THEN GOTO 1380 ELSE CALL SOUND(100,
329,1,391,1,466,1):: SC=SC+50 :: DISPLAY
 AT(1,8)SIZE(9):SC
1370 CALL SPRITE(#6,118,13,Z,S,22,0):: C
ALL HCHAR(Z/8+1,S/8+1,32)
1380 FOR I=1 TO 2 :: Z=Z+4 :: CALL LOCAT
E(#1,Z+8,S,#2,Z,S,#3,Z,S,#4,Z+8,S):: NEX
T I :: CALL DELSPRITE(#6):: RETURN
1390 CALL CLEAR :: CALL SCREEN(5):: DISP
LAY AT(4,3):"J U M P I N G  M I N E R":
:"h h h h h h h h h h h h h h":: : : :"
1985 VON WILLI DOELTSCH"
1400 DISPLAY AT(15,4):"PREPARE FOR STATI
ON";LEV :: DISPLAY AT(20,6):LIFE;" CHANC
E LEFT"
1410 IF SC>0 THEN GOTO 1460
1420 RESTORE 250
1430 READ I :: IF I=99 THEN GOTO 1420 EL
SE IF I<110 THEN READ II ELSE II=I :: I=
1
1440 CALL SOUND(INT(200*I),II,0,II+3,0,I
I+6,0)
1450 CALL KEY(1,TA,ST):: IF TA<>18 THEN
GOTO 1430
1460 FOR I=1 TO 3 :: CALL SCREEN(13):: C
ALL SOUND(-1500,880,1,440,18,884,20):: F
OR A=1 TO 200 :: NEXT A
1470 CALL SCREEN(12):: CALL SOUND(-1500,
110,1,220,18,111,25):: FOR A=1 TO 200 ::
 NEXT A :: NEXT I :: CALL CLEAR :: CALL
SCREEN(2):: RETURN
1480 IF SC<=NSC THEN GOTO 1490 ELSE NSC=
SC
1490 CALL CLEAR :: DISPLAY AT(9,7):"HIGH
SCORE =";NSC :: DISPLAY AT(13,7):"SCORE
    =";SC
1500 DISPLAY AT(22,1):"FUER NEUES SPIEL
ODER ENDE": :"BITTE DIE TASTEN J ODER N"
 :: SC,TOTAL,LIFE,TI=0
1510 RESTORE 290
1520 READ I :: IF I=99 THEN GOTO 1510 EL
SE IF I<110 THEN READ II ELSE II=I :: I=
1
1530 CALL SOUND(INT(120*I),II,0,II+3,0,I
I+6,0)
1540 CALL KEY(3,TA,ST):: IF TA=74 THEN G
OTO 710 ELSE IF TA=78 THEN CALL CLEAR ::
 END ELSE GOTO 1520
```

Ende des Listings

Enemy-Attack

Alles Gute kommt von oben!?!

Stutzen Sie den Fallschirmspringern die Flügel!

Ein absolutes Knüllerspiel auf dem Apple in Maschinensprache! Ein Ballerspiel vom Feinsten! Jeder Schuß ein Treffer – oder auch nicht. Sie können sich zu Beginn entscheiden, ob Sie Ihre Geschosse steuern wollen oder nicht. Entschließen Sie sich, die Kügelchen zu steuern, da Sie dann die Hubschrauber mit Ihren Geschossen jagen können.

Die eigentliche Gefahr geht jedoch von den Fallschirmspringern aus. Sobald links oder rechts von der Kanone vier von den Faltern den Boden erreicht haben, wird Ihr Feuerstützpunkt in die Luft gesprengt. Von den Hubschraubern selbst geht keine Gefahr aus. Allerdings bringen diese mehr Punkte als die Jäger am Schirm. Natürlich müssen Sie auch Opfer bringen, um zu Punkten zu gelangen. Jeder Schuß kostet Sie einen Punkt. Bei Dauerfeuer können Sie sich ja ausrechnen, wieviel Ihnen übrig bleiben (null). Nehmen Sie sich unbedingt vor den Männchen in Acht, die versuchen, auf Ihrer Plattform zu landen. Wenn ihnen das gelingt, sind Sie ebenfalls hinüber. Die dritte (und damit letzte) Todesform sind Bomben, die in einem Zwischenlevel von Star-Fightern auf Sie geworfen werden. Bis zu diesem Zeitpunkt sollten Sie schon ein paar Punkte unter Dach und Fach gebracht haben, da diese Bomben unbedingt getroffen werden müssen. Dazu empfiehlt sich Dauerfeuer!

Begeben Sie sich nun auf die High-Score-Jagd bei *Enemy-Attack!*

Zur Eingabe von Enemy-Attack:

Mit ***CALL-151*** in den Monitor gehen. Dann das Listing von 1D1F-552F eingeben. Danach mit ***CTRL-C*** und ***Return*** den Monitor verlassen und das Programm mit ***BSAVE ENEMY-ATTACK,A$1D1F,L$3810*** abspeichern und mit ***BRUN ENEMY-ATTACK*** starten.

Das Listing ➡

```
aaaaaaaaaaaai";
<KD> 1200 RETURN
<MI> 1210 DATA 0,24,24,24,24,255,255,24,
24,24,24,24,24,24,60,126,126,255
```

The End

Square-Painter Akkord-Malerei

Ein echtes Geschicklichkeitsspiel von feinster Güte! Quadrate scheinen sich für diese Art von Spielen wohl am besten zu eignen, wenn wir an das Spiel Quadrato aus der letzten Ausgabe zurückdenken.

In diesem Spiel geht es darum, mit dem eigenen (zu steuernden) Quadrat anderen auszuweichen. Gleichzeitig müssen Flächen bemalt und ein bestimmtes Ziel-Quadrat erreicht werden.
Der Schwierigkeitsgrad steigt auf verschiedene Art und Weise:
1. Die bewegten Quadrate sind zu Beginn nur winzig, beginnen jedoch, pulsierend zu wachsen und wieder zu schrumpfen. Nach jedem dritten „Puls" steigt das bis dahin größte Ausmaß um eine Einheit, bis maximal zum Achtfachen der ursprünglichen Größe. Die Anzahl der Pulswiederholungen zwischen den Wachstumsschritten kann im Editmodus verändert werden. Natürlich wird die wachsende Schwierigkeit auch durch Punkte abgegolten.
2. Die Geschwindigkeit wächst bei jeder Runde. (Die Anfangsgeschwindigkeit kann ebenfalls im Editmodus verändert werden. Sollte man auch bereit sein, eine Verlangsamung des Gesamtspielablaufs in Kauf zu nehmen, kann man in den Zeilen 940 und 1010 die „!" löschen, und bekommt dafür eine akustische Begleitung.).
Achtung: Die ALPHA-LOCK-Taste muß während des Spielens gelöst sein!

Das Listing:

```
100 !******************
110 !* SQUARE-PAINTER *
120 !* -------------- *
130 !*   TI 99/4A +:  *
140 !* EXTENDET BASIC *
150 !* -------------- *
160 !* [ABSTAKT GAME] *
170 !* -------------- *
180 !*   (c) 1984     *
190 !*  GRABUSCHNIG   *
200 !* NEUTORSTR. 55  *
210 !* 5020 SALZBURG  *
220 !******************
230 !
240 !
250 RANDOMIZE
260 GOTO 290
270 V=9 :: H=2 :: S=H :: FOR I=0 TO 14 :
: CALL COLOR(I,V,H):: NEXT I :: CALL SCR
EEN(S):: ACCEPT A
280 V=16 :: H=5 :: S=H :: FOR I=0 TO 14
:: CALL COLOR(I,V,H):: NEXT I :: CALL SC
REEN(S):: ACCEPT A
290 GOTO 450 :: DIM M1(9),M2(9),MU(12),D
A(12),T(60),MU1(8),MU2(8),PAT(8),VF(8),H
F(8),VL(8),HL(8),TC(16),VS(8),HS(8)
300 ZAE,HI,G,PA,QU,X,Y,K,ST,VZ,HZ,PU,S,A
,B,I,I2,I3,I4,I5,I6,I7,I8,I9,J,J1,J5,J4,
J3,J2,J6,J7,J8=J9
310 GES,WID,TM,W,I1,PL,V,H,PR,RU,C1=C2
320 Z$=W$
330 CALL CHAR :: CALL SPRITE :: CALL LOC
ATE :: CALL CLEAR :: CALL SCREEN :: CALL
 HCHAR :: CALL VCHAR :: CALL GCHAR
340 CALL MOTION :: CALL COINC :: CALL DE
LSPRITE :: CALL PATTERN :: CALL MAGNIFY
:: CALL COLOR :: CALL KEY :: CALL JOYST
:: CALL SOUND
350 DATA FF8080808080808080808080808080F
FFF0101010101010101010101010101FF
360 !@P-
370 DATA 007F4040404040404040404040407F0
000FE0202020202020202020202020202FE
380 DATA 00003F2020202020202020202020203F000
00000FC0404040404040404040404FC
390 DATA 0000001F1010101010101010101F00000
0000000F8080808080808080808F8
400 DATA 000000000F0808080808080F0000000
000000000F0101010101010F
410 DATA 00000000000704040404070000000000
00000000000E020202020E
420 DATA 0000000000000302020300000000000
0000000000000C04040C
430 DATA 000000000000000101000000000000
000000000000000808
440 DATA 80808080808080FF
450 CALL CLEAR :: CALL COLOR(1,2,2)
460 FOR I1=96 TO 128 STEP 4 :: READ Z$ :
: CALL CHAR(I1,Z$):: NEXT I1 :: CALL CHA
R(32,Z$,136,Z$)
470 !
480 ! VARIABLENBELEGUNG
490 !
500 CALL MAGNIFY(4)
510 GOSUB 1270 :: DISPLAY AT(12,1):"S Q
U A R E - P A I N T E R"
520 GOSUB 2030
530 GOSUB 1870
540 GOSUB 1790
550 GOSUB 2040
```

```
560 V.H.VZ.HZ=1
570 GOSUB 1770
580 GOSUB 2050
590 J=124 :: FOR I=1 TO 8 :: PAT(I)=J ::
 J=J-4 :: NEXT I ! PATTERN-VAR
600 J=1 :: FOR I=1 TO 8 :: VL(I),HL(I)=J
 :: J=J+32 :: NEXT I ! LOCATE-VAR
610 FOR I=1 TO 6 :: VS(I)=I+1 :: NEXT I
620 FOR I=1 TO 8 :: HS(I)=I+7 :: NEXT I
! SPR.COINC-VAR
630 J=1 :: FOR I=1 TO 8 :: VF(I),HF(I)=J
 :: J=J+4 :: NEXT I ! #1-FELD-(756)
640 J=16 :: FOR I=1 TO 8 :: TC(I)=J :: J
=J+2 :: NEXT I ! TOL.-VAR
650 GOSUB 2120
660 ! EINSTELLUNGS-                    MOEGLÍ
CHKEITEN:
670 GES=2 !ANFANGSGESCHWINDIGKEIT
680 QU=3 !ANZAHL DER QUADRATE DES SPIELE
RS
690 WID=3 ! ANZAHL DER PULSWIDERHOLUNGEN
700 RU=1 :: PR.PU=0
710 CALL CLEAR :: CALL SCREEN(8)
720 DISPLAY AT(19,7):"G A M E   O V E R"
730 DISPLAY AT(21.11):"NEW GAME? "
740 DISPLAY AT(23,9):"[FEUERTASTE]"
750 CALL KEY(2,K,ST):: IF K<>18 THEN 750
760 PL.PR=0 :: CALL CLEAR :: CALL COLOR(
1,16,2)
770 CALL SCREEN(4):: GOSUB 1170
780 GOSUB 1310
790 !
800 !
810 ! VORPROGRAMM
820 !
830 DISPLAY AT(11.7):"CUONTDOWN:"
840 FOR ZAE=56 TO 1 STEP -2 :: CALL SOUN
D(-150,880.10.1760.10):: DISPLAY AT(11.1
8):ZAE :: CALL KEY(2,K,ST):: CALL SOUND(
-150,T(ZAE),0)
850 CALL JOYST(2,X,Y):: IF K=18 OR X OR
Y THEN 920
860 CALL SOUND(-150,880.10.1760.10):: DI
SPLAY AT(11.18):ZAE-1 :: CALL KEY(2,K,ST
):: CALL SOUND(-150,T(ZAE-1),0)
870 CALL JOYST(2.X,Y):: IF K=18 OR X OR
Y THEN 920
880 NEXT ZAE :: CALL SOUND(1000.147.0)
890 !
900 ! HAUPTPROGRAMM
910 !
920 CALL HCHAR(11,8,32,16):: I6=2
930 I6=MIN(I6+1.8):: FOR J3=1 TO WID ::
FOR I=1 TO I6 :: PA=PAT(I)
940 !CALL SOUND(-4000,MU1(I).19,MU2(I),1
9)
950 CALL PATTERN(#15,PA.#2.PA.#3,PA.#4.P
A,#5,PA.#6,PA.#7,PA,#8.PA.#9,PA,#10,PA,#
11,PA,#12,PA.#13,PA,#14.PA)
960 CALL COINC(#1,#VS(V).TC(I),C1):: CAL
L COINC(#1,#HS(H),TC(I),C2):: IF C1 OR C
2 THEN GOTO 1080 :: ELSE CALL KEY(2,K,ST
):: CALL JOYST(2.X.Y)
970 IF K=18 THEN 1490 ELSE IF X XOR Y TH
EN IF X THEN IF X=4 THEN H=MIN(8,H+1)ELS
E H=MAX(1,H-1)ELSE IF Y=4 THEN V=MAX(1,V
-1)ELSE V=MIN(6,V+1)ELSE 990
980 CALL LOCATE(#1.VL(V).HL(H))
990 NEXT I
1000 FOR I=I6 TO 1 STEP -1 :: PA=PAT(I)
1010 !CALL SOUND(-4000,MU1(I),19,MU2(I),
19)
1020 CALL PATTERN(#15,PA.#2.PA.#3.PA.#4.
PA,#5.PA,#6,PA.#7,PA,#7,PA,#8,PA.#9,PA,#
10,PA,#11,PA,#12.PA.#13.PA,#14.PA)
1030 CALL COINC(#1,#VS(V),TC(I),C1):: CA
LL COINC(#1,#HS(H),TC(I).C2):: IF C1 OR
C2 THEN GOTO 1080 :: ELSE CALL KEY(2,K,S
T):: CALL JOYST(2,X,Y)
1040 IF K=18 THEN 1550 ELSE IF X XOR Y T
HEN IF X THEN IF X=4 THEN H=MIN(8,H+1)EL
SE H=MAX(1,H-1)ELSE IF Y=4 THEN V=MAX(1,
V-1)ELSE V=MIN(6,V+1)ELSE 1060
1050 CALL LOCATE(#1,VL(V),HL(H))
1060 NEXT I :: NEXT J3 :: GOTO 930
1070 ! KOLLISION
1080 FOR I=8 TO 1 :: CALL PATTERN(#1,PAT
(I)):: NEXT I
1090 CALL MOTION(#2,0.0,#3,0.0.#4,0,0,#5
,0,0)
1100 CALL MOTION(#6,0.0,#7,0.0.#8.0,0,#9
,0.0,#10,0,0,#11.0,0.#12,0,0,#13,0,0.#14
,0.0,#15,0.0)
1110 QU=QU-1 :: GOSUB 1910 :: CALL CLEAR
 :: CALL COLOR(1,2,2)
1120 CALL DELSPRITE(ALL):: GOSUB 1660
1130 IF QU>0 THEN 760 ELSE 670
1140 !
1150 ! SPRITES SETZEN
1160 !
1170 S=GES+RU :: CALL DELSPRITE(ALL)
1180 J=1 :: FOR I=2 TO 7 :: CALL SPRITE(
#I,124,8,J,INT(RND*192)+1,0,S):: J=J+32
:: S=-S :: NEXT I :: J=1 :: FOR I=8 TO 1
5 :: CALL SPRITE(#I,124,8,INT(RND*256)+1
,J,S,0)
1190 S=-S :: J=J+32 :: NEXT I :: RETURN
1200 !
1210 ! BILDSCHIRMAUFBAU
1220 !
1230 !
1240 !
1250 ! FARBGEBUNG
1260 !
1270 CALL COLOR(1,2,2,13.16,10.14,16,14.
2,3,2.3,3.2,4,3,2,5,3.2,6,3,2,7,3,2,8,3,
2):: RETURN
1280 !
1290 ! SPRITE #1 UND ZIELBERECHNUNG
1300 !
1310 ON INT(RND*8)+1 GOTO 1320,1330,1340
,1350,1360,1370,1380,1390
1320 V.H=1 :: VZ=6 :: HZ=INT(RND*8)+1 ::
 GOTO 1400
```

```
1330 V,H=1 :: VZ=INT(RND*6)+1 :: HZ=8 ::
 GOTO 1400
1340 V=1 :: H=8 :: VZ=6 :: HZ=INT(RND*8)
+1 :: GOTO 1400
1350 V=1 :: H=8 :: VZ=INT(RND*6)+1 :: HZ
=1 :: GOTO 1400
1360 V=6 :: VZ,H=1 :: HZ=INT(RND*8)+1 ::
 GOTO 1400
1370 V=6 :: H=1 :: HZ=8 :: VZ=INT(RND*6)
+1 :: GOTO 1400
1380 V=6 :: H=8 :: HZ=1 :: VZ=INT(RND*6)
+1 :: GOTO 1400
1390 V=6 :: H=8 :: VZ=1 :: HZ=INT(RND*8)
+1 :: GOTO 1400
1400 CALL SPRITE(#1,96,6,VL(V),HL(H)):
FOR I4=VF(VZ)TO VF(VZ)+3 :: CALL HCHAR(I
4,HF(HZ),136,4):: NEXT I4
1410 A=VF(VZ):: B=HF(HZ):: RETURN
1420 !
1430 ! LOESCH-ZIEL
1440 !
1450 FOR I4=A TO A+3 :: CALL HCHAR(I4,B,
32,4):: NEXT I4 :: RETURN
1460 !
1470 ! GELB 1
1480 !
1490 FOR I7=100 TO 124 STEP 4 :: CALL PA
TTERN(#1,I7):: NEXT I7 :: CALL GCHAR(VF(
V),HF(H),G):: IF G>40 THEN IF G=128 THEN
 1510 ELSE 1610
1500 FOR I8=VF(V)TO VF(V)+3 :: CALL HCHA
R(I8,HF(H),128,4):: NEXT I8
1510 FOR I9=120 TO 96 STEP -4 :: CALL PA
TTERN(#1,I9):: NEXT I9 :: PL=PL+1 :: GOT
O 990
1520 !
1530 ! GELB 2
1540 !
1550 FOR I7=100 TO 124 STEP 4 :: CALL PA
TTERN(#1,I7):: NEXT I7 :: CALL GCHAR(VF(
V),HF(H),G):: IF G>40 THEN IF G=128 THEN
 1570 ELSE 1610
1560 FOR I8=VF(V)TO VF(V)+3 :: CALL HCHA
R(I8,HF(H),128,4):: NEXT I8
1570 FOR I9=120 TO 96 STEP -4 :: CALL PA
TTERN(#1,I9):: NEXT I9 :: PL=PL+1 :: GOT
O 1060
1580 !
1590 ! ZIEL
1600 !
1610 GOSUB 1450 :: PR=INT((100*PL)/.48)/
100 :: PU=INT((RU*PR*100*I6)-(RU*ZAE*100
)+PU)
1620 CALL DELSPRITE(ALL):: GOSUB 2080 ::
 CALL CLEAR :: CALL COLOR(1,2,2):: GOSUB
 1660 :: RU=RU+1 :: S=GES+RU :: GOTO 760
1630 !
1640 ! KOMMUNIKATION
1650 !
1660 CALL SCREEN(14)
1670 HI=MAX(HI,PU):: DISPLAY AT(7,2):"RE
KORD:";HI
1680 DISPLAY AT(9,2):"PUNKTE:";PU
1690 DISPLAY AT(11,2):"PROZENT D. FLAECH
E:";PR
1700 DISPLAY AT(13,2):"COUNTDOWN:";ZAE
1710 DISPLAY AT(15,2):"RUNDE:";RU
1720 DISPLAY AT(17,2):"QUADRATE:";QU
1730 DISPLAY AT(19,2):"     [ FEUERTASTE
]"
1740 FOR J8=1 TO 200 :: NEXT J8
1750 CALL KEY(2,K,ST):: IF K=18 THEN RET
URN ELSE 1750
1760 ! DATEN SPIELMUS.
1770 RESTORE 1780 :: FOR I=1 TO 8 :: REA
D MU1(I),MU2(I):: NEXT I :: RETURN
1780 DATA 294,415,311,440,330,466,349,49
4,370,523,392,554,415,587,440,622
1790 ! COUNTDOWNMELODIE
1800 RESTORE 1810 :: FOR I=56 TO 0 STEP
-1 :: READ T(I):: NEXT I :: RETURN
1810 DATA 587,659,698,587,659,698,784,65
9,698,784,880,698,784,880,932,784,880
1820 DATA 698,784,659,698,587,659,554,58
7
1830 DATA 440,466,392,440,349,392,330,34
9,294,392,330,349
1840 DATA 294,330,277,294,220,233,196,22
0,175,196,165,175,147,196,165,175
1850 DATA 147,165,139,147
1860 !STARTMELODIE
1870 TM=0 :: RESTORE 1880 :: FOR W=1 TO
3 :: FOR I=1 TO 11 :: READ MU(I),DA(I)::
 NEXT I :: FOR I=1 TO 11 :: CALL SOUND(8
0*DA(I),MU(I),2):: NEXT I :: GOSUB 2040
:: NEXT W
1880 DATA 880,1,784,1,880,8,30000,8,784,
1,698,1,659,1,587,1,554,5,587,12,30000,1
2
1890 DATA 440,1,392,1,440,8,30000,4,330,
4,349,4,277,4,294,12,30000,12,30000,-1,3
0000,-1
1900 DATA 220,1,196,1,220,8,30000,8,196,
1,175,1,165,1,147,1,139,5,147,12,30000,6
1910 !
1920 ! MELODIE BEI KOLLISION
1930 !
1940 CALL SOUND(640,139,2)
1950 CALL SOUND(640,139,2,165,2)
1960 CALL SOUND(640,139,2,165,2,196,2)
1970 CALL SOUND(960,139,2,196,2,233,2)
1980 CALL SOUND(800,147,0,196,0,220,0)
1990 CALL SOUND(800,147,1,165,1,220,1)
2000 CALL SOUND(1440,147,2,185-TM,2,220,
2)
2010 TM=10 :: RETURN
2020 !
2030 FOR I=1 TO 8 :: CALL SPRITE(#I,92+4
*I,16,120,112):: NEXT I
2040 A=SGN(RND-.5)*(RND*20+30):: B=SGN(R
ND-.5)*(RND*20+10):: FOR I=1 TO 8 :: CAL
L MOTION(#I,A,B):: NEXT I :: RETURN
2050 ! DATEN F. SIEGMELODIE
2060 RESTORE 2070 :: FOR I=1 TO 9 :: REA
```

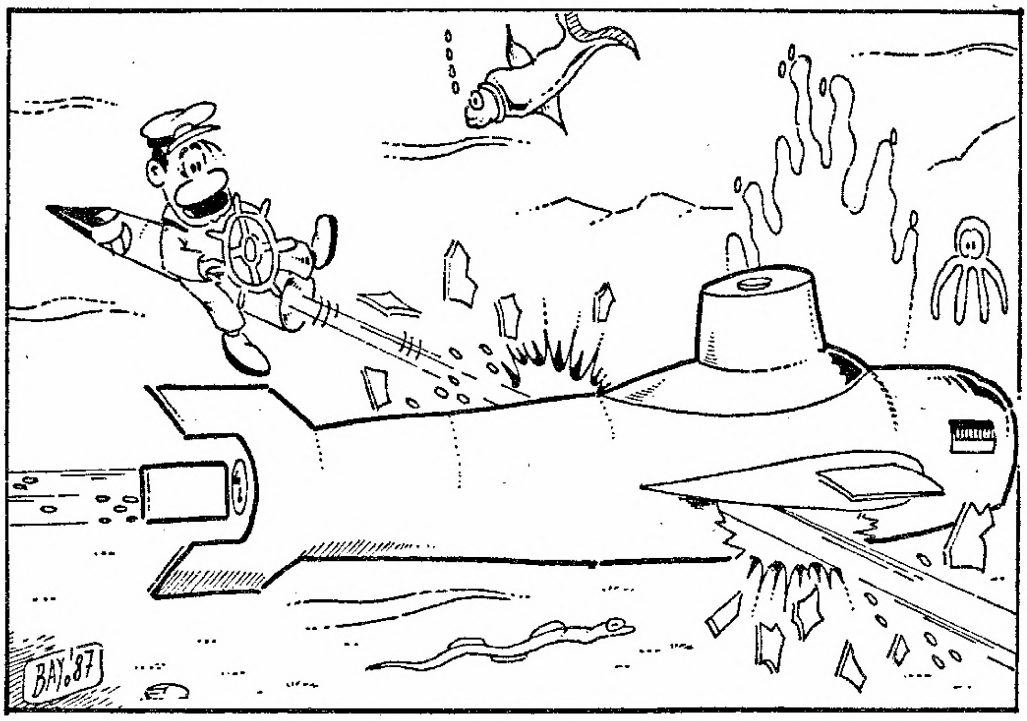
```
D M1(I),M2(I):: NEXT I :: RETURN
2070 DATA 30000,294,277,330,294,277,330,
349,349,392,392,330,440,440,440,466,440,
392
2080 ! SIEGMELODIE
2090 I=1 :: FOR I1=1 TO 6 STEP 2 :: CALL
 SOUND(240,M1(I1),2):: CALL SOUND(240,M1
(I1+1),2)
2100 FOR I3=1 TO 3
2110 FOR I2=I TO I+2 :: CALL SOUND(80,M2
(I2),2):: NEXT I2 :: NEXT I3 :: I=I+3 ::
 NEXT I1 :: CALL SOUND(480,440,2):: RETU
RN
2120 ! SPIELBESCHREIBUNG
2130 CALL DELSPRITE(ALL):: CALL SCREEN(2
):: CALL CLEAR :: PRINT "SPIELBESCHREIBU
NG ?[J/N]"
2140 CALL KEY(0,A,B):: IF A=106 OR A=74
THEN 2150 ELSE IF A=78 OR A=110 THEN RET
URN ELSE 2140
2150 RESTORE 2220
2160 FOR I=1 TO 10 :: READ Z$ :: IF Z$="
OOO" THEN 2200 ELSE PRINT :Z$
2170 NEXT I
2180 FOR I=180 TO 0 STEP -1 :: DISPLAY A
T(24,20):"[";I;"]"
2190 NEXT I :: GOTO 2160
2200 FOR I=150 TO 0 STEP -1 :: DISPLAY A
T(24,20):"[";I;"]"
2210 NEXT I :: RETURN
2220 DATA 1.SPIELSTEUERUNG:,DAS SPIELER-
QUADRAT (DUN=,KELBLAU) WIRD IN VIER,RICH
TUNGEN MIT JOYSTICK [2].GESTEUERT.
2230 DATA DURCH DRUECKEN AUF DIE,FEUERTA
STE (MALTASTE!) WIRD,EINE FLAECHE DES SP
IEL=.FELDES ROT GEFAERBT ODER,DAS ZIEL-Q
UADRAT GELOESCHT.
2240 DATA 2.AUFGABEN DES SPIELERS:,MOEGL
ICHST LANGES WARTEN.WAEHREND DES COUNTDO
WNS -,SONST PUNKTEABZUG;,VERMEIDUNG VON
ZUSAMMEN=
2250 DATA STOESSEN MIT BEWEGTEN,QUADRATE
N;,EINFAERBEN DER VOM SPIELER-,QUADRAT B
ESETZTEN FLAECHE:,VIOLETTES ZIEL-QUADRAT
,LOESCHEN;
2260 DATA 3.BEWERTUNG :,BEWERTET WIRD DE
R PROZENT=.ANTEIL DER GEFAERBTEN.FLEACHE
 VON DER GESAMT=,FLAECHE. ALLERDINGS ERS
T AB
2270 DATA LOESCHUNG DES ZIEL-QUADRATES.O
OO
```

Ende des Listings

„Sub-Attack"

Geschicklichkeit und Ballerei

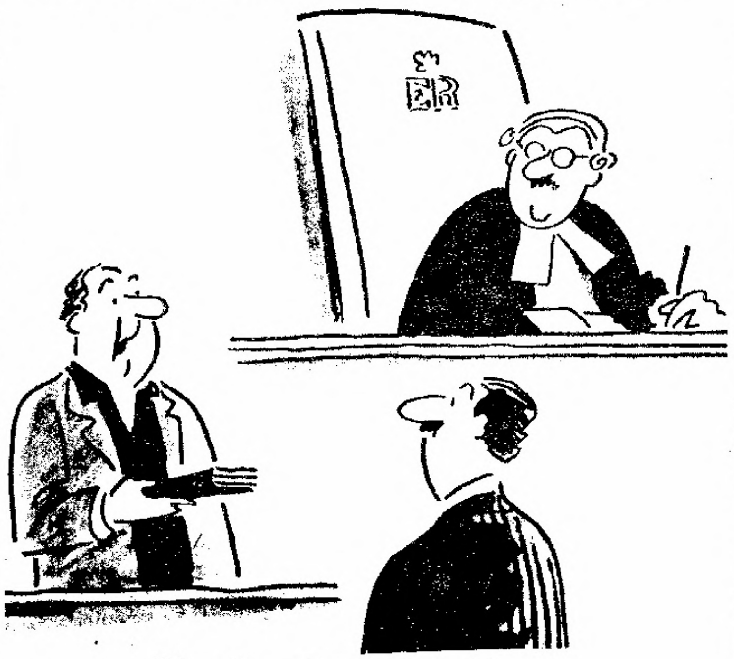
Als Pilot eines neuen Untersee-Flitzers hat es Sie bei einer Routinefahrt in eine Meeresgrotte verschlagen. Hier erwarten Sie seltsame und gefährliche Kreaturen. Kommt Ihr U-Boot in Kontakt mit einem dieser Wesen, so ist es um Sie geschehen!



Zögern Sie deshalb nicht, von den Torpedos Gebrauch zu machen. Damit haben Sie eine längere Überlebenschance! Während Sie die Aqua-Smurfs noch abschießen können, funktioniert dies mit den Tiefsee-Gnulps nicht mehr. Hier sind schnelle Ausweichmanöver lebenswichtig.
Natürlich müssen Sie auch jede Kollision mit den Höhlenwänden vermeiden, denn nach dem dritten Schiff ist endgültig Schluß!

Steuerung:
Die Spielsteuerung kann sowohl mit dem Paddle/Joystick, als auch mit der Tastatur erfolgen. Nach dem Start wird Paddle-Steuerung vorausgesetzt. Ist kein Paddle (oder Joystick) angeschlossen, so kann mit CTRL-K auf Tastatur-Steuerung umgeschaltet werden. Dann können folgende Tasten benutzt werden:
J – bewegt das U-Boot nach links
L – nach rechts

```
<OG> 6660 DATA 173,40,6,201,255,240,6,14
1
<FJ> 6670 DATA 7,210,206,40,6,173,38,6
<EH> 6680 DATA 201,120,208,13,169,16,141
,40
<IP> 6690 DATA 6,169,0,141,1,210,76,177
<HN> 6700 DATA 187,201,121,208,13,169,16
,141
<EP> 6710 DATA 40,6,169,0,141,3,210,76
<MJ> 6720 DATA 4,188,169,0,133,77,174,42
<GE> 6730 DATA 6,173,48,6,157,0,99,202
<NG> 6740 DATA 189,0,99,141,48,6,169,155
<JG> 6750 DATA 157,0,99,206,42,6,174,43
<IJ> 6760 DATA 6,173,49,6,157,0,120,202
<PM> 6770 DATA 189,0,120,141,49,6,169,15
6
<LM> 6780 DATA 157,0,120,206,43,6,174,44
<IF> 6790 DATA 6,173,50,6,157,0,130,202
<PA> 6800 DATA 189,0,130,141,50,6,169,15
7
<LJ> 6810 DATA 157,0,130,206,44,6,174,45
<IK> 6820 DATA 6,173,51,6,157,0,137,232
<PM> 6830 DATA 189,0,137,141,51,6,169,15
8
<MK> 6840 DATA 157,0,137,238,45,6,174,46
<IF> 6850 DATA 6,173,52,6,157,0,110,232
<PI> 6860 DATA 189,0,110,141,52,6,169,15
9
<MG> 6870 DATA 157,0,110,238,46,6,174,47
<IP> 6880 DATA 6,173,53,6,157,0,125,232
<AA> 6890 DATA 189,0,125,141,53,6,169,15
7
<GB> 6900 DATA 157,0,125,238,47,6,76,4
<LG> 6910 DATA 30,162,0,160,80,169,0,157
<DA> 6920 DATA 0,78,157,0,79,157,0,80
<FK> 6930 DATA 157,0,81,157,0,82,157,0
<CA> 6940 DATA 86,152,157,0,83,200,232,2
24
<FF> 6950 DATA 40,208,226,169,0,133,112,
169
<CE> 6960 DATA 87,133,113,169,0,160,23,1
45
<ND> 6970 DATA 112,200,145,112,230,113,1
65,113
<FJ> 6980 DATA 201,143,208,239,76,0,183,
224
<DB> 6990 DATA 2,227,2,78,140,168,140
```



Ende des Listings

Biathlon

Eiskalte Jungs schweißgebadet!

Da der Winter in deutschen Landen kein Ende zu nehmen scheint, sollen auch unsere TI- Freunde in den Spielgenuß von Biathlon kommen.

Klemmen Sie den Joystick in Port 1, und geben Sie den Namen und die Nationalität des Spielers ein, nachdem das Programm gestartet hat.

Jetzt kann's losgehen. Sie sind auf der linken Bildschirmseite. Durch rechts-links-Bewegungen des Joysticks schiebt sich Ihre Spielfigur nach rechts. Haben Sie den rechten Bildschirmrand erreicht, kommen Sie ins nächste Bild. Die gesamte Route beinhaltet fünf verschieden Bilder, die insgesamt dreimal durchfahren werden müssen. Die sehr gute Grafik und der sehr schnelle Aufbau der verschiedenen Screens zeichnen dieses Programm aus. Im wesentlichen haben Sie bei diesem Spiel mit zwei Schwierigkeiten zu kämpfen. Zunächst müssen Sie den richtigen Rhythmus der Bewegungen finden, da es nichts nützt, den Joystick wie ein Wilder von rechts nach links zu fetzen. Desweiteren müssen Sie blitzschnell umschalten können, wenn Sie keuchend an den Schießstand kommen. Jeder Fehlschuß kostet Sie nämlich zehn Sekunden.

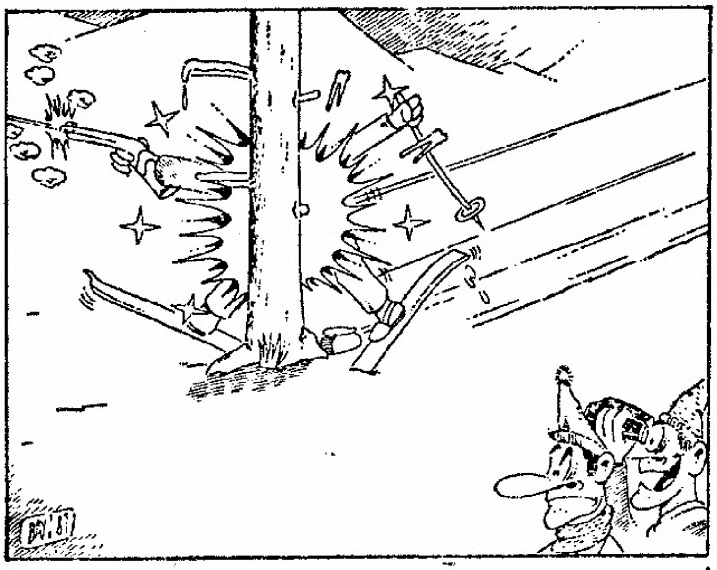
Die Jagd auf die Bestzeit kann also beginnen.

Steuerung:

Joystick links/rechts - Bewegen des Spielers

Joystick hoch - Schuß laden

Joystick runter - bücken (Abfahrt), Schuß



„Immerhin hat der alte Angeber Recht behalten: Er hat mit geschlossenen Augen ins Schwarze getroffen!"

Das Listing

```
100 ! B I A T H L O N
110 ! HOLGER SCHAEFER                    BERLIN
ER STR.18                  6082 MOERFELDEN
120 O=1 :: DIM A(30)
130 HIM=4 :: HITI=60
140 CALL CLEAR :: CALL SCREEN(2):: CALL
MAGNIFY(3)
150 FOR C=3 TO 8 :: CALL COLOR(C,16,2)::
 NEXT C
160 DISPLAY AT(9,8):"B I A T H L O N" ::
 DISPLAY AT(20,12):"TC SOFT" :: FOR P=1
TO 250 :: NEXT P :: CALL CLEAR :: GOTO 4
50
170 DATA FFFFFFFFFFFFFFFF,FF819989898981
FF,8080808080808080,0101010101010101
180 DATA 00000001070F0F1F,1F1F0F0F070100
00,000000C0F0F8F8FC,FCFCF8F8F0C00000
190 DATA 00000000FF0000FF,00030C30C30C30
C0,C30C30C000000000,0000000000030C30
200 DATA 00000000C0300CC3,300C0300000000
00,C0300CC3300C0300,8040201008040201
210 DATA 001C22262A32221C,00081808080808
1C,001C22020C10203E,003E02040C02221C
220 DATA 00040C14243E0404,003E203C020222
1C,000E10203C22221C,003E020408101010
230 DATA 001C22221C22221C,001C22221E0204
38,0000100000100000,00FFFFFFFFFFFFFF
240 DATA C78301010183C7FF,00000000000000
00,0000000000000000,0000000000000000
250 DATA FFFFFFFFFFFFFFFF,183C7E66667E66
66,7C7E66667C66667E,3C7E666060667E3C
260 DATA 787C6E66666E7C78,7E7E607C7C607E
7E,7E7E607C7C606060,3C7E66606E667E3C
270 DATA 6666667E7E666666,18181818181818
18,7E7E0606066E7C38,666E6C787C6E6666
280 DATA 6060606060607E7E,63777F7F6B6363
63,63737B7F6F676363,3C7E666666667E3C
290 DATA 7C7E667E7C606060,3C7E66666E667E
3D,7C7E667E7C6C6666,3C7E66300C667E3C
300 DATA 7E7E181818181818,6666666666667E
3C,6363637777363E1C,6363636B6B7F7F36
310 DATA 6363763E3E776363,666666663C3C18
18,7E7E060C18307E7E,FFFFFFFFFFFFFF00
320 DATA 000000000305080D,08050300000000
00,0000000080402060,2040800000000000
330 DATA 181818187EA92400,181838540A0800
00,1010385400000000,1818183C5E952402
340 DATA 000000071F3F3F7F,FFFF7F7F3F0F07
00,00000000E0FCFEFE,FFFFFEFEF8E0C000
350 DATA FFFFFFFFFFFFFFFF,FFFCF0C0000000
00,FFFFFFFFFFFCF0C0,FF7F3F1F0F070301
360 DATA 0303030103030303,070A12244F0404
FF,80C08000C0B08888,88884841FE0010E0
370 DATA 18183C3C3C7E7EFF,01010303070F3F
7F,8080C0C0E0F0FCFE,FFFFFFFFFFFFFFFF
380 DATA 01010303070F1F3F,8080C0C0E0F0F8
FC,0101030307070F1F,8080C0C0E0E0F0F8
390 DATA 0102040810204080,FF000000000000
00,C020100804030C31,030C1020FF8080FF
400 DATA 0303030103030307,0B122404FF0000
0F,80C0800080C0A098,84848454E44441FE
410 DATA 0303030103030303,060A14270C30C3
04,80C08000E0908884,A0A1C34C70C00000
420 DATA 0101010001010305,081000030C0003
0C,C0E0C080C0F0C8C4,CEF1C48CB0C00000
430 DATA 030303011F031F03,1A0D631A060100
00,80C0800080C0E080,808080601887601C
440 DATA 0000041209050303,1A06631A060100
00,70787020F0F0D0F0,808080601887601C
450 RESTORE 170 :: FOR Z=32 TO 143 :: RE
AD Z$ :: CALL CHAR(Z,Z$):: NEXT Z
460 CALL COLOR(1,2,16,2,15,16,9,14,16,10
,6,16,11,3,16,12,15,16)
470 B$(1)="hhhhhhhhhhhhhhhhhhhhhhhhhhhhh
hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
hhhhhhhhhhhhhhh@@@@@@@@@@@@@@@@@@@@@@@@
@@@"
480 B$(2)="@@p@@@@@@p@@p@@@@@@@p@@@@@@@@
vsw@p@@vswvsw@p@@@vsw@@@@p@@vswvsw@tsuvs
wvsw@@tsu@p@vsw@tsuvsw@tsutsuvsw@@tsuvsw
vsw"
490 B$(3)="@qsrtsu@qsrtsutsu@@qsrtsutsu@
qsrqsr@qsrqsrqsr@@qsrqsrqsr@@c@@`@@@c@@`
@@c@@@@`@@c@@`@@@@@@@@@@@@@@@@@@@@@@@@@
@@@"
500 B$(4)="((((((((((((((((((((((((((((@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@"
510 C$(1)="hhhhhhhhhhhhhhhhhhhhhhhhhhhhh
hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
hhhhhhhhhhhhhhhhhhhhhhhhhji/@@@@@@@@@@@@
@@@"
520 C$(2)="hhhhhhhhji@p@/@@@@@@@@@@@@@@h
hhhhhjip@@b@@z((((((((((((hhhhji@@b@@@+
)*/@@@@@@@@@@@@hhji@@@@@@+)*@@@/@@@@@@@p
@@@"
530 C$(3)="ji@@@@@@+)*@p@@@@/@@@@@vsw@@@
@@@@@+)*@@vsw@@@@/@@@@tsu@@@@@@+)*@@@@ts
```

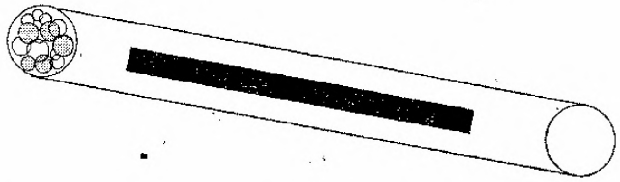
```
u@@@@@/@p@tsu@@@@+)*@@@@@@qsr@@@@@@/a@qs
r@@"
540 C$(4)="+)*@@@@@@@@@c@@@@@@@@/@@`@@@*
@@@@@@@@@@@@@@@@@@@@@/@@@@@@@@@@@@@@@@@@
@@@@@@@@@@/@@@@@@@@@@@@@@@@@@@@@@@@@@@@@/
@@@"
550 D$(1)="hhhhhhhhhhhhhhhhhhhhhhhhhhhhh
hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
hhh"
560 D$(2)="@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@p@@@@@p@@@p@@@@@@p@@@@p@@@@vsw@@@vsw@v
sw@p@@@b@@pvsw@@@vsw@@@tsu@tsuvsw@@@@@av
sw@"
570 D$(3)="@@tsu@@@qsr@@a@tsu@@@@@@tsu@@
@qsr@@@qsr@@@@tsu@@@@@@qsr@@@qsr@@@@c@@@
@@qsr@@@@@@@c@@@@@`@@@@@@@@@@@@`@@@@@@@@
@@@"
580 D$(4)="((((((((((((((((((((((((((((@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@"
590 E$(1)="hhhhhhhhhhhhhhhhhhhhhhhhhhhhh
hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
hhhhhhhhhhhhhhh,@@@@@@@@@@@@@@@@@@@@khhhh
hhh"
600 E$(2)="-.,@@@@@@@@@@@@@@p@@@@khhhhhh@
@-.,@@@@@@@@@@@@b@@@@@khhhhh@@@@-.,@@@p@@
@@@@@p@@@@khhhh@@@@@@-.,@a@p@@@@@b@@@@xy
yyy"
610 E$(3)="@@@p@@@p-.,@a@@@@@@@@@x@@@@@@
@vsw@vsw@-.,@@@@@@@@x@@@@@@@@tsu@vsw@@@-
.,@@@@@x@@@@@@@@@tsu@tsu@@@@@-.,@@x@@@@@
@@@"
620 E$(4)="@@qsr@tsu@@@@@@@-.((((((((((@
@qsr@qsr@@@@@@@@x@@@@@@@@@@@@@`@@@c@@@@@
@@@x@@@@@@@@@@@@@@@@@@@@@@@@@@x@@@@@@@@@
@@@"
630 F$(2)="@@@@@@@@@@@[[[[![[[[@@@@@@@@@@p
@@p@@p@@p#@@@@@@#@p@@p@@p@@b@@b@@b@@b#@@
@@@@#@b@@b@@b@@@@@@@@@@@@@#@@@@@@#@@@@@@@
@@@"
640 F$(3)="@@@@@@@@@@x;;;;;;;/@@@@@@@@@@@
@@@@@@@@x@@@@@@@@@@/@@@@@@@@@@@@@@@@@@x@@@@
@@@@@@@@/@@@@@@@@@@@@@@@@x@@@@@@@@@@@@@@/@@@
@@@"
650 F$(4)="@@@@@@x@@@@@@@@@@@@@@@@@/@@@@@@
@@@@x@@@@@@@@@@@@@@@@@@@/@@@@@@@@yy@y@y@y@
y@y@y@y@y@yy@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@" :: CALL CLEAR
660 ! SPIELBEGINN
670 DISPLAY AT(18,1):"NAME DES ATHLETEN:
" :: ACCEPT AT(20,1)SIZE(15)VALIDATE(UAL
PHA):NAM$(A(O)):: IF NAM$(A(O))="" THEN
670
680 DISPLAY AT(22,1):"NATION:" :: ACCEPT
 AT(22,10)SIZE(3)VALIDATE(UALPHA):NAT$(A
(O)):: IF NAT$(A(O))="" THEN 680
690 CALL HCHAR(1,1,32,704)
700 U=10 :: BILD=1 :: TI,M=0 :: DISPLAY
AT(22,25)SIZE(4):"0:00"
710 DISPLAY AT(2,7):"B I A T H L O N" ::
 DISPLAY AT(20,11):"TC SOFT" :: DISPLAY
AT(22,1)SIZE(15):NAM$(A(O)):: DISPLAY AT
(22,18)SIZE(3):NAT$(A(O))
720 ON BILD GOTO 880,910,940,970,1000
730 FOR W=1 TO 500 :: NEXT W :: CALL SPR
ITE(#1,P,2,H,20):: U=U+8 :: GOSUB 1480
740 ON BILD GOTO 890,920,950,980,1010
750 IF M<=HIM THEN 760 ELSE 810
760 IF TI<=HITI THEN 780 ELSE 810
770 FOR Z=1 TO 300 :: NEXT Z
780 HIM=M :: HITI=TI :: DISPLAY AT(24,1)
SIZE(15):NAM$(A(O)):: DISPLAY AT(24,18)S
IZE(3):NAT$(A(O)):: DISPLAY AT(24,25)SIZ
E(1):USING "#":M
790 DISPLAY AT(24,26)SIZE(1):":" :: IF T
I<10 THEN 800 ELSE DISPLAY AT(24,27)SIZE
(2):USING "##":TI :: GOTO 810
800 DISPLAY AT(24,27)SIZE(2):USING "0#":
TI
810 FOR Z=1 TO 1000 :: NEXT Z :: DISPLAY
 AT(22,1)SIZE(28):"IST NOCH EIN ATHLET A
M START"
820 CALL KEY(0,K,S):: IF K=74 OR K=106 T
HEN 850
830 IF K=78 OR K=110 THEN 860
840 GOTO 820
850 RU=0 :: CALL DELSPRITE(ALL):: CALL H
CHAR(1,1,32,704):: O=O+1 :: GOTO 670
860 CALL CLEAR :: CALL DELSPRITE(ALL)::
END
870 ! UNTERPROGRAMME
880 DISPLAY AT(4,1):B$(1):B$(2):B$(3):B$
(4):: P=108 :: H=113 :: GOTO 730
890 GOTO 1080
900 CALL DELSPRITE(#1):: BILD=BILD+1 ::
GOTO 720
910 DISPLAY AT(4,1):C$(1):C$(2):C$(3):C$
(4):: P=132 :: H=113 :: GOTO 730
920 GOTO 1140
930 CALL DELSPRITE(#1):: BILD=BILD+1 ::
GOTO 720
940 DISPLAY AT(4,1):D$(1):D$(2):D$(3):D$
(4):: P=108 :: H=113 :: GOTO 730
950 GOTO 1080
960 CALL DELSPRITE(#1):: BILD=BILD+1 ::
GOTO 720
970 DISPLAY AT(4,1):E$(1):E$(2):E$(3):E$
(4):: P=136 :: H=47 :: GOTO 730
980 GOTO 1210
990 CALL DELSPRITE(#1):: BILD=BILD+1 ::
GOTO 720
1000 DISPLAY AT(4,1):D$(1):F$(2):F$(3):F
$(4):: H=1 :: P=136 :: GOTO 730
1010 GOTO 1340
1020 CALL DELSPRITE(#1):: BILD=1 :: RU=R
U+1 :: IF RU=3 THEN 750
1030 GOTO 720
1040 IF TI<10 THEN 1070 :: IF TI=60 THEN
 1050 :: DISPLAY AT(22,27)SIZE(2):USING
"##":TI :: RETURN
1050 TI=0 :: M=M+1 :: IF M>9 THEN 1060 :
: DISPLAY AT(22,25)SIZE(1):USING "#":M :
```

```
: DISPLAY AT(22,27)SIZE(2):"00" :: RETUR
N
1060 DISPLAY AT(22,24)SIZE(2):USING "##"
:M :: DISPLAY AT(22,27)SIZE(2):"00" :: R
ETURN
1070 DISPLAY AT(22,27)SIZE(2):USING "0#"
:TI :: RETURN
1080 CALL JOYST(1,X,Y):: IF X=4 THEN 112
0 ELSE GOSUB 1100 :: GOTO 1080
1090 CALL JOYST(1,X,Y):: IF X=-4 THEN 11
30 ELSE GOSUB 1100 :: GOTO 1090
1100 TI=TI+1 :: GOSUB 1040 :: FOR Z=1 TO
 60 :: NEXT Z :: RETURN
1110 CALL POSITION(#1,X,Y):: IF Y>225 TH
EN 900 :: RETURN
1120 CALL MOTION(#1,0,13):: CALL PATTERN
(#1,124):: CALL SOUND(160,123,22,-5,23):
: FOR Z=1 TO 35 :: NEXT Z :: CALL MOTION
(#1,0,0):: GOSUB 1100 :: GOSUB 1110 :: G
OTO 1090
1130 CALL MOTION(#1,0,13):: CALL PATTERN
(#1,108):: CALL SOUND(160,123,22,-5,23):
: FOR Z=1 TO 35 :: NEXT Z :: CALL MOTION
(#1,0,0):: GOSUB 1100 :: GOSUB 1110 :: G
OTO 1080
1140 CALL JOYST(1,X,Y):: IF X=4 THEN 118
0 ELSE GOSUB 1160 :: GOTO 1140
1150 CALL JOYST(1,X,Y):: IF X=-4 THEN 11
90 ELSE GOSUB 1160 :: GOTO 1150
1160 TI=TI+1 :: FOR Z=1 TO 60 :: NEXT Z
:: GOSUB 1040 :: RETURN
1170 CALL POSITION(#1,X,Y):: IF Y>121 TH
EN 1200 :: RETURN
1180 CALL MOTION(#1,-4,8):: CALL PATTERN
(#1,128):: CALL SOUND(100,123,22,-5,18):
: FOR Z=1 TO 34 :: NEXT Z :: CALL MOTION
(#1,0,0):: GOSUB 1160 :: GOSUB 1170 :: G
OTO 1150
1190 CALL MOTION(#1,-4,8):: CALL PATTERN
(#1,132):: CALL SOUND(100,123,22,-5,18):
: FOR Z=1 TO 34 :: NEXT Z :: CALL MOTION
(#1,0,0):: GOSUB 1160 :: GOSUB 1170 :: G
OTO 1140
1200 CALL LOCATE(#1,57,128):: CALL PATTE
RN(#1,108):: GOTO 1080
1210 CALL MOTION(#1,4,8):: L=0
1220 CALL JOYST(1,X,Y):: IF Y=-4 THEN 12
50
1230 L=L+1 :: IF L=2 THEN GOSUB 1310 ELS
E 1320
1240 GOTO 1220
1250 CALL MOTION(#1,8,16):: CALL PATTERN
(#1,140):: GOTO 1270
1260 CALL POSITION(#1,X,Y):: IF Y>146 TH
EN 1290 :: RETURN
1270 L=L+1 :: IF L=2 THEN GOSUB 1300 ELS
E 1330
1280 GOTO 1270
1290 CALL MOTION(#1,0,0):: CALL LOCATE(#
1,113,152):: CALL PATTERN(#1,108):: GOTO
 1080
1300 L=0 :: TI=TI+1 :: GOSUB 1040 :: RET
URN
1310 L=0 :: TI=TI+1 :: GOSUB 1040 :: RET
URN
1320 FOR Z=1 TO 37 :: NEXT Z :: GOSUB 12
60 :: GOTO 1220
1330 FOR Z=1 TO 23 :: NEXT Z :: GOSUB 12
60 :: GOTO 1270
1340 CALL HCHAR(10,15,60,5):: HO=101
1350 FOR S=3 TO 8 :: HO=HO+8 :: IF S=8 T
HEN 1450
1360 CALL JOYST(1,X,Y):: IF Y=4 THEN 137
0 ELSE GOSUB 1440 :: GOTO 1360
1370 CALL SPRITE(#2,92,15,50,HO,17+RU,0)
1380 CALL JOYST(1,X,Y):: IF Y=-4 THEN CA
LL MOTION(#2,0,0):: GOTO 1400 ELSE BILD=
BILD-1 :: IF BILD=1 THEN 1460
1390 CALL POSITION(#2,X,Y):: IF X>78 THE
N 1370 ELSE 1380
1400 CALL SOUND(83,110,0,-7,10):: CALL P
OSITION(#2,X,Y):: IF X>65 AND X<72 THEN
1430 ELSE 1410
1410 CALL DELSPRITE(#2):: FOR I=1 TO 10
:: TI=TI+1 :: GOSUB 1040 :: NEXT I
1420 NEXT S
1430 FOR Z=1 TO 70 :: NEXT Z :: CALL SOU
ND(83,147,7):: CALL DELSPRITE(#2):: CALL
 HCHAR(10,12+S,64):: GOTO 1420
1440 TI=TI+1 :: GOSUB 1040 :: FOR Z=1 TO
 80 :: NEXT Z :: RETURN
1450 GOTO 1020
1460 BILD=5 :: TI=TI+1 :: GOSUB 1040 ::
GOTO 1390
1470 ! SONNE,WOLKE
1480 CALL SPRITE(#11,36,11,30,U)
1490 CALL SPRITE(#10,100,15,31,U+(U*.4)+
10)
1500 RETURN
```

Ende des Listings

Snackimacy

Der Pillenknick



Als ein kleiner Verwandter von Pac-Man, mit dem Namen Snackimacy, befinden Sie sich in einem Paradies von Pillen. Paradies, weil Sie Ihre Artgenossen so richtig auf den Geschmack gebracht haben. Zur Zeit gibt es für Sie einfach nichts Köstlicheres als Pillen. Kein Wunder, daß Sie sich wie ein Wilder auf die kleinen runden Dinger stürzen, und Sie bis auf die Letzte vertilgen wollen.

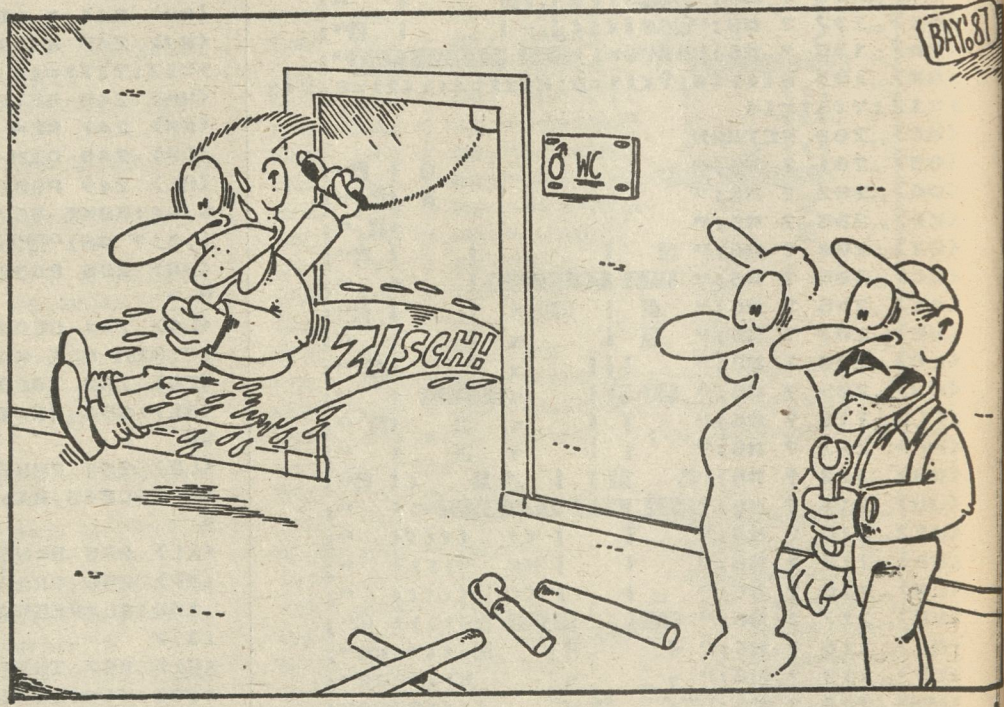
Willy in the City

Da klappern die Toiletten...

Der tapfere Bergmann aus dem Superoldie MANIC MINER ist da, denn es ist dem Programmierer tatsächlich gelungen, eine erstaunlich gute Konvertierung für den Ti mit Extended Basic zu erstellen.Werden Sie alle Gefahren überwinden können?
Als Bergmann hat man es ganz schön schwer, denn Willy hat mal wieder seine ganze Kohle in der Kneipe gelassen. Was bleibt ihm also anderes übrig, als einen kleinen Nebenjob anzufangen? Als Straßenkehrer muß er nun den städtischen Müll aufsammeln. Erst wenn er in einer Straße alles aufgesammelt hat, darf er weitergehen, aber das ist wahrlich eine schwere Aufgabe. Die Gemäuer dieser Stadt sind nämlich ganz schön brüchig, und nur zu schnell bricht der kleine Willy durch. Außerdem gibt es noch jede Menge üble Gestalten wie zum Beispiel Kloschüsseln, fliegende Gartenscheren und andere Seltsamkeiten, die unserem Willy nach dem Leben trachten. Da hilft nur noch vorsichtiges Steuern, denn sonst verliert Willy sehr schnell eines seiner Leben. Hat er aber trotz aller Schwierigkeiten einen Tag überlebt, so gibt's ein Zusatzleben und das Game kann weitergehen. Springen kann unser Held mit dem Feuerknopf, gesteuert wird mit dem Joystick an Port 1.

Zur Eingabe:

Einfach eingeben und absaven. Gestartet wird mit RUN.



„Vielleicht haben wir die Spülung doch etwas zu stark eingestellt?"

Das Listing:

```
100 ! ********************
110 ! *                  *
120 ! *   PETER'S  SOFT  *
130 ! *                  *
140 ! * PROUDLY  PRESENTS *
150 ! *                  *
160 ! ********************
170 ! *                  *
180 ! * WILLY IN THE CITY *
190 ! *                  *
200 ! *     VER 1.2      *
210 ! *                  *
220 ! ********************
230 ! *                  *
240 ! *   PETER  HLAWNA  *
250 ! *     AM KOBEL 8   *
260 ! *   82  ROSENHEIM  *
270 ! *                  *
280 ! ********************
290 ! *                  *
300 ! *   (C) 1987  BY   *
310 ! *                  *
320 ! *   PETER'S SOFT   *
330 ! *                  *
340 ! ********************
350 !
360 CALL CLEAR :: CALL SCREEN(2):: CALL
MAGNIFY(3)
370 ON ERROR 2360
380 !
390 ! ZEICHENDATAS
400 !
410 DATA 3C24247E2424183C,26C5CD3E6C6C36
64,,
420 DATA 3C24247E2424183C,2424243C181818
38,,
430 DATA 3C24247E2424183C,64A3B37C36366C
26,,
440 DATA 3C24247E2424183C,2424243C181818
1C,,
450 FOR I=96 TO 111 :: READ A$ :: CALL C
```

```
HAR(I,A$):: NEXT I
460 CALL CHAR(120,RPT$("AA55",16))
470 CALL CHAR(44,"")
480 CALL HCHAR(1,1,30,768)
490 FOR I=3 TO 7 :: CALL COLOR(I,16,1)::
 NEXT I
500 SC,RU,P=0 :: Q=2 :: L=5
510 GOSUB 1350 :: GOSUB 1370
520 !
530 ! ROUTINE
540 !
550 RU=RU+1 :: P=P+100 :: R=0 :: IF RU>5
 THEN RU=1 :: L=L+1 :: GOSUB 1370
560 FOR I=3 TO 22 :: CALL HCHAR(I,3,44,2
8):: NEXT I
570 ON RU GOSUB 1420,1640,1820,2000,2170
580 RESTORE 2400 :: FOR I=1 TO 36 :: REA
D A,B :: CALL SOUND(A*1.4,B,5,B/2,5)
590 CALL JOYST(1,X,Y):: IF X THEN 610
600 NEXT I :: GOTO 580
610 CALL MOTION(#11,0,D1,#12,0,D1*D2,#13
,0,D1*D3)
620 FOR CO=1 TO 5
630 CALL POSITION(#1,Y,X):: CALL GCHAR((
Y+23)/8,(X+7)/8,A)
640 IF A=40 OR A=35 THEN 670 ELSE IF A=3
0 OR A>45 THEN 1090
650 IF A=44 THEN 900
660 CALL HCHAR((Y+23)/8,(X+7)/8,A+1)
670 CALL JOYST(1,XJ,YJ):: CALL MOTION(#1
,0,XJ):: CALL KEY(1,K,S):: IF K=18 THEN
780
680 IF XJ THEN Q=-Q :: CALL PATTERN(#1,1
04+XJ+Q)
690 CALL COINC(ALL,A):: IF A THEN 990
700 CALL JOYST(1,XJ,YJ):: CALL MOTION(#1
,0,XJ):: CALL KEY(1,K,S):: IF K=18 THEN
780
710 IF XJ THEN Q=-Q :: CALL PATTERN(#1,1
04+XJ+Q)
720 NEXT CO
730 R=R+1 :: CALL POSITION(#11,Y,X):: IF
 X<=X1 THEN D=D1 ELSE IF X>=X2 THEN D=-D
1
740 CALL MOTION(#11,0,D,#12,0,D*D2,#13,0
,D*D3):: GOTO 620
750 !
760 ! SPRUNG
770 !
780 FOR I=-8+(XJ=0)TO 8-(XJ=0)STEP 2 ::
CALL MOTION(#1,I,XJ)
790 CALL COINC(ALL,A):: IF A THEN 990
800 Q=-Q :: IF XJ THEN CALL PATTERN(#1,1
04+XJ+Q)
810 CALL POSITION(#1,Y,B):: CALL GCHAR((
Y+24)/8,(B+7)/8,A):: IF A=33 OR A>45 THE
N 1090
820 IF I>=0 AND A<>44 THEN 850
830 IF XJ THEN Q=-Q :: CALL PATTERN(#1,1
04+XJ+Q)
840 NEXT I :: GOTO 900
850 CALL MOTION(#1,0,0):: CALL LOCATE(#1
,INT((Y+4)/8)*8+1,B+XJ/4):: CALL SOUND(-
10,110,5)
860 GOTO 730
870 !
880 ! FALL
890 !
900 B,K=0
910 CALL MOTION(#1,9,0):: CALL COINC(ALL
,A):: IF A THEN 990
920 CALL POSITION(#1,Y,X):: CALL GCHAR((
Y+24)/8,(X+7)/8,A)
930 IF A=44 THEN B=B+1 :: GOTO 910
940 CALL MOTION(#1,0,0):: CALL LOCATE(#1
,INT((Y+4)/8)*8+1,X)
950 IF A>45 OR B>8 THEN 1090 ELSE CALL S
OUND(-10,110,5):: GOTO 730
960 !
970 ! GETROFFEN
980 !
990 CALL MOTION(#1,0,0,#11,0,0,#12,0,0,#
13,0,0)
1000 FOR J=2 TO 10 :: CALL COINC(#1,#J,1
2,A):: IF A=0 THEN 1040
1010 CALL SOUND(-100,550,5)
1020 CALL DELSPRITE(#J):: CALL MOTION(#1
1,0,D,#12,0,D*D2,#13,0,D*D3):: N=N-1
1030 SC=SC+P :: GOSUB 1350 :: IF K=18 TH
EN 840 ELSE 720
1040 NEXT J :: FOR J=11 TO 13 :: CALL CO
INC(#1,#J,16,A):: IF A THEN 1090
1050 NEXT J :: FOR J=14 TO 19 :: CALL CO
INC(#1,#J,8,A):: IF A THEN 1090
1060 NEXT J :: CALL COINC(#1,#20,16,A)::
 IF A AND N=0 THEN 1250 ELSE 1090
1070 IF K=18 THEN 840 ELSE 720
1080 !
1090 ! TOT
1100 !
1110 CALL MOTION(#1,0,0,#11,0,0,#12,0,0,
#13,0,0)
1120 RESTORE 2400 :: FOR I=1 TO 6 :: REA
D A,B :: CALL SOUND(A*1.8,B,5,B/2,5,B/1.
5,5):: NEXT I
1130 L=L-1 :: IF L=0 THEN 1160
1140 GOSUB 1370 :: GOTO 570
1150 !
1160 ! GAME OVER
1170 !
1180 DISPLAY AT(22,1):"      G A M E   O
 V E R"
1190 RESTORE 2400
1200 FOR I=1 TO 36 :: READ A,B :: CALL S
OUND(A*1.5,B,5,B/2,5):: CALL KEY(1,K,S):
: IF K=18 THEN CALL DELSPRITE(ALL):: GOT
O 500
1210 NEXT I :: DISPLAY AT(22,1):"  PRESS
 FIRE FOR NEW GAME" :: GOTO 1190
1220 !
```

```
1230 ! FELD GESCHAFFT
1240 !
1250 FOR I=1 TO 2 :: FOR J=880 TO 440 ST
EP -110 :: CALL SOUND(-100,J,5,J/2,5)::
NEXT J :: NEXT I
1260 FOR I=880 TO 220 STEP -110 :: CALL
SOUND(-100,I,5,I/2,5):: NEXT I
1270 DISPLAY AT(22,1):"BONUS:"
1280 A=MAX((100-R)*P/20,0):: FOR I=1 TO
300 :: NEXT I :: DISPLAY AT(22,8):USING
"####":A
1290 FOR I=0 TO 30 STEP 2 :: CALL SOUND(
-100,440,I,880,I):: NEXT I
1300 SC=SC+A :: GOSUB 1350
1310 CALL DELSPRITE(ALL):: CALL COLOR(1,
2,2,2,2,2,8,2,2):: GOTO 530
1320 !
1330 ! UNTERPROGRAMME
1340 !
1350 HI=MAX(HI,SC):: DISPLAY AT(2,1):USI
NG "SCORE: ######   HI : ######":SC,HI
1360 RETURN
1370 DISPLAY AT(23,1)SIZE(L*2):RPT$("ln"
,MIN(L-1,14))&"nn" :: DISPLAY AT(24,1)SI
ZE(L*2):RPT$("mo",MIN(L-1,14))&"oo"
1380 RETURN
1390 !
1400 ! FELDER
1410 !
1420 CALL CHAR(124,"000000003E3E3E3E3F3F
3F3F0F1F1F3F00000000020408000F8F8F8F8F0
8080")
1430 CALL CHAR(128,"00000000105438FE3854
10"&RPT$("0",42))
1440 CALL CHAR(132,"00000000000000000001
0905150B0F07000000000000000000204050A0
E0C0")
1450 CALL CHAR(40,"FFFFAA55AAFF0000FFFFA
A55AA00000000FFFF55AA000000000000FFAA")
1460 CALL CHAR(35,"FFFFFFFFFFFFFFFF",46,
"FFFFAA55AAFF0000FFFFFFFFFFFFFFFF",90,"F
F202020FF040404")
1470 DISPLAY AT(6,1):"((()))))))))))))(#()
))))))))((("
1480 DISPLAY AT(11,1):"((()))))(#(((((.(
((((()))))((("
1490 DISPLAY AT(16,1):"))))))))((.)))))))
)(#(((,,((("
1500 DISPLAY AT(22,1):"THE HOUSE IN THE
MAIN STREET"
1510 CALL VCHAR(7,18,90,4):: CALL VCHAR(
12,12,90,4):: CALL VCHAR(17,22,90,4)
1520 CALL HCHAR(20,23,40,3):: CALL HCHAR
(18,23,40):: CALL HCHAR(15,10,40,2):: CA
LL HCHAR(13,11,40)
1530 CALL HCHAR(10,17,40):: CALL HCHAR(8
,17,40):: CALL HCHAR(21,3,35,28)
1540 CALL HCHAR(18,3,41,2):: CALL HCHAR(
19,5,41,2):: CALL HCHAR(21,22,47)
1550 CALL COLOR(0,5,5,1,7,2,2,7,4,8,16,7
,10,16,5)
1560 CALL SPRITE(#20,120,3,145,17)
1570 CALL SPRITE(#1,100,16,145,225)
1580 CALL SPRITE(#11,124,16,25,41,#12,12
4,2,25,121,#13,124,16,145,139):: D,D1=2
:: D2=1 :: D3=-1
1590 CALL SPRITE(#2,128,14,13,65,#3,128,
3,13,185,#4,128,5,69,17)
1600 CALL SPRITE(#5,128,2,69,233,#6,128,
15,85,33,#7,128,6,85,205)
1610 CALL SPRITE(#8,128,8,109,124,#9,128
,10,125,89,#10,128,15,125,145)
1620 N=9 :: X1=42 :: X2=121
1630 RETURN
1640 CALL CHAR(124,"00000000387C66637F32
7F63667C380000000000000000000FCFFFC")
1650 CALL CHAR(128,"000000000022777F081C
3E")
1660 CALL CHAR(132,"0000000101033B1F0703
06040C080000000000000080B8F0C080C0406020
")
1670 CALL CHAR(40,"FFFFFFFFFFFFFFFFFFFFA
A55AA55AA550000FFAA55AA55AA0000000000FF5
5AA")
1680 CALL VCHAR(7,16,41,14):: CALL VCHAR
(7,17,41,14)
1690 DISPLAY AT(8,1):",,,,,,,,,,((,)),((
,,,,,,,,,,,,,,,,,,,,(,,,)),,,(,,,,,,,,,,,
,,,,,,,,,,)),,,,,,,,,,,,,,,,,,,,,((,,,)
),,,((,,,,,,,,,"
1700 DISPLAY AT(12,1):",,,,,,,,,,,,,,)),,
,,,,,,,,,,,,,,,,,,,(,,,)),,,(,,,,,,,,,,,
,),),,,,,((,)),((,,,,,),),,,),,,),,,,,,,
)),,,,,,,),,,),"
1710 CALL HCHAR(21,3,40,28):: CALL VCHAR
(16,9,41,5):: CALL VCHAR(16,24,41,5)
1720 DISPLAY AT(22,1):"          THE GARD
EN"
1730 CALL COLOR(0,3,3,2,13,4,10,13,3)
1740 CALL SPRITE(#1,100,3,145,225)
1750 CALL SPRITE(#11,124,8,41,81,#12,124
,8,113,161,#13,124,8,145,81)
1760 CALL SPRITE(#2,128,16,13,124,#3,128
,15,77,124,#4,128,8,100,25,#5,128,12,100
,225)
1770 CALL SPRITE(#14,132,11,53,65,#15,13
2,12,53,177,#16,132,12,69,108,#17,132,12
,69,134)
1780 CALL SPRITE(#18,132,12,89,121,#20,1
20,11,145,17)
1790 D,D1=2 :: D2=-1 :: D3=1
1800 N=4 :: X1=81 :: X2=161
1810 RETURN
1820 CALL CHAR(124,"00030606030000010308
0C3F7F675B1800B84C4CF8E0C0E0F0989CFCFEE6
5A18")
1830 CALL CHAR(128,"000010103878DC3660")
1840 CALL CHAR(132,"00010905150B0F070000
00000000000000000204050A0E0C00")
1850 CALL CHAR(40,"FFFFFFFF00000000EEEEE
```

```
E000000000000EEEE00000000000000EE")
1860 FOR I=9 TO 17 STEP 4 :: DISPLAY AT(
I,1):"(((,,,)))))))))))))))),,,(((
1870 CALL HCHAR(I-2,7,41):: CALL HCHAR(I
-2,26,41):: NEXT I
1880 CALL HCHAR(19,7,41):: CALL HCHAR(19
,26,41)
1890 DISPLAY AT(5,7)SIZE(16):")),,)),,,,
)),,))"
1900 CALL HCHAR(21,3,40,28):: DISPLAY AT
(22,1):"        THE MAIN STREET"
1910 CALL COLOR(0,15,15,2,16,2,10,2,15)
1920 CALL SPRITE(#1,100,15,145,225)
1930 CALL SPRITE(#11,124,6,49,81,#12,124
,9,81,161,#13,124,6,113,81)
1940 CALL SPRITE(#2,128,11,21,125,#3,128
,11,69,125)
1950 CALL SPRITE(#4,128,11,101,17,#5,128
,11,101,233,#6,128,11,133,125)
1960 CALL SPRITE(#14,132,11,153,109,#15,
132,11,153,133)
1970 CALL SPRITE(#20,120,3,17,225)
1980 D,D1=2 :: D2=-1 :: D3=1 :: X1=81 ::
 X2=161 :: N=5
1990 RETURN
2000 CALL CHAR(124,"C0C0E1E3F7F7FFFFFFFF
F7F7E3E1C0C078FCFEFAFBFFFFEFF0FFFFFFFEFE
FC78")
2010 CALL CHAR(128,"00000000181818242424
3C")
2020 CALL CHAR(40,"FFFFFFFFFFFFFFFFAA55A
A55AA55AA550000AA55AA55AA55000000000055A
A55")
2030 FOR I=18 TO 20 :: CALL HCHAR(I,10,4
1,14):: NEXT I
2040 DISPLAY AT(6,1):",,,,,,,,,,,((((((,
,,,,,,,,,,,,,,,,,,((,,,//,,,((,,,,,,,,,,
,,,((,,,,,,//,,,,,,((,,,,,,,,,,,,,,,,(((/
/(((,,,,,,,,,,"
2050 DISPLAY AT(10,1):",,((,,,,,,,,,//,,
,,,,,,,((,,,,,,,,,,,,,,,//,,,,,,,,,,,,,(
(,,,,,,,(((((//(((((,,,,,,((,,,,,,,,,,,,
//,,,,,,,,,,,,"
2060 DISPLAY AT(14,1):",,(((,,,,,,,,(//(,
,,,,,,,(((,,,,,,,,,,,,,,,//,,,,,,,,,,,,,
,,,,,,,,,,((//((,,,,,,,,,,,,,,,,,,,,,,,,/
///,,,,,,,,,,,,"
2070 DISPLAY AT(18,1):",,,,,(()))((////(
(()))((,,,,,,,,,,//))))))))))))))))//,,,,,,
,,((//))))))))))))))))//((,,,((////((((((
((((((((////((("
2080 DISPLAY AT(22,1):"    AT THE BIG FO
UNTAIN"
2090 CALL COLOR(0,6,6,2,8,5,10,5,6)
2100 CALL SPRITE(#1,108,6,145,25)
2110 CALL SPRITE(#11,124,6,73,105,#12,12
4,6,145,89,#13,124,6,145,169)
2120 CALL SPRITE(#2,128,11,14,125,#3,128
,11,71,65,#4,128,11,71,185)
2130 CALL SPRITE(#5,128,11,93,41,#6,128,
11,93,209,#7,128,11,149,125)
2140 CALL SPRITE(#14,132,3,33,115,#15,13
2,3,33,127,#20,120,16,17,225)
2150 D,D1=2 :: D2,D3=-.5 :: N=7 :: X1=10
5 :: X2=137 :: N=6
2160 RETURN
2170 CALL CHAR(124,"0F3F7F7F152A7F7F3F0F
000000000000E0F8FCFC50A8FCFCF8E0")
2180 CALL CHAR(128,"000000001C223E222222
1C")
2190 CALL CHAR(40,"FFFFFFFFFFFFFFFFFFFFA
A55AA55AA550000FF55AA55AA550000000000FFA
A55")
2200 CALL CHAR(35,"FFFFFFFFFFFFFFFF")
2210 DISPLAY AT(6,1):",,,,,,))))),,,,,,,,
))))),,,,,,,,,,,,))),,))),,,,,,))),,,,,,,
,,))),,,,))),,,,))),,,,))),,,,,,,))),,,,,,)),
,))),,,,,,))),,,"
2220 DISPLAY AT(10,1):",,,))),,,,,,))),,))
,,,,,,))),,,,,))),,,,,,,,))))),,,,),,,))),,,
,))),,,,,,,,))))),,,))),),))),,,))),,,))))),,,
))),,,)))))),,))),"
2230 DISPLAY AT(14,1):",))),,))))))),,))),,
))))))),,))),,))),,######,,))),,######,,))),,
))),,))))))),,))),,[[[[[[[,,))),,))),,,))))),,,
))),,,[[[[,,,)),"
2240 DISPLAY AT(18,1):",))),,,,,,,,,,))),,
,[[[[,,,))),,))),,,,,,,,,(,,(,,,,,,,,,))),(
(((,,,,,,,,,/,,/,,,,,,,,,((((////((((((((/
///((((((((////"
2250 DISPLAY AT(22,1):"  THE JUNK FOOD R
ESTAURANT"
2260 CALL COLOR(0,2,2,1,7,1,2,12,9,8,7,7
,10,16,2)
2270 CALL SPRITE(#1,100,16,137,233)
2280 CALL SPRITE(#11,124,11,60,121,#12,1
24,11,95,73,#13,124,11,95,169)
2290 CALL SPRITE(#2,128,16,13,77,#3,128,
16,13,173,#4,128,16,29,125,#5,128,16,53,
25)
2300 CALL SPRITE(#6,128,16,53,225,#7,128
,16,101,77,#8,128,16,101,173,#20,120,15,
145,121)
2310 D,D1=2 :: D2,D3=-1 :: X1=105 :: X2=
137 :: N=7
2320 RETURN
2330 !
2340 ! ERROR
2350 !
2360 CALL MOTION(#1,0,0):: ON ERROR 2360
 :: A=44 :: RETURN NEXT
2370 !
2380 ! MUSIKDATAS
2390 !
2400 DATA 200,330,200,349,200,392,200,34
9,400,330,400,262
2410 DATA 200,330,200,349,200,392,200,34
9,200,330,200,349,200,392
2420 DATA 200,440,200,466,200,440,200,46
6,200,440,400,392
```

```
2430 DATA 200,415,200,392,200,349,200,39
2
2440 DATA 100,415,100,392,100,349,100,39
2,400,415
2450 DATA 100,392,100,349,100,311,100,34
9,200,392,200,262,700,523
```

Ende des Listings

Tetrade

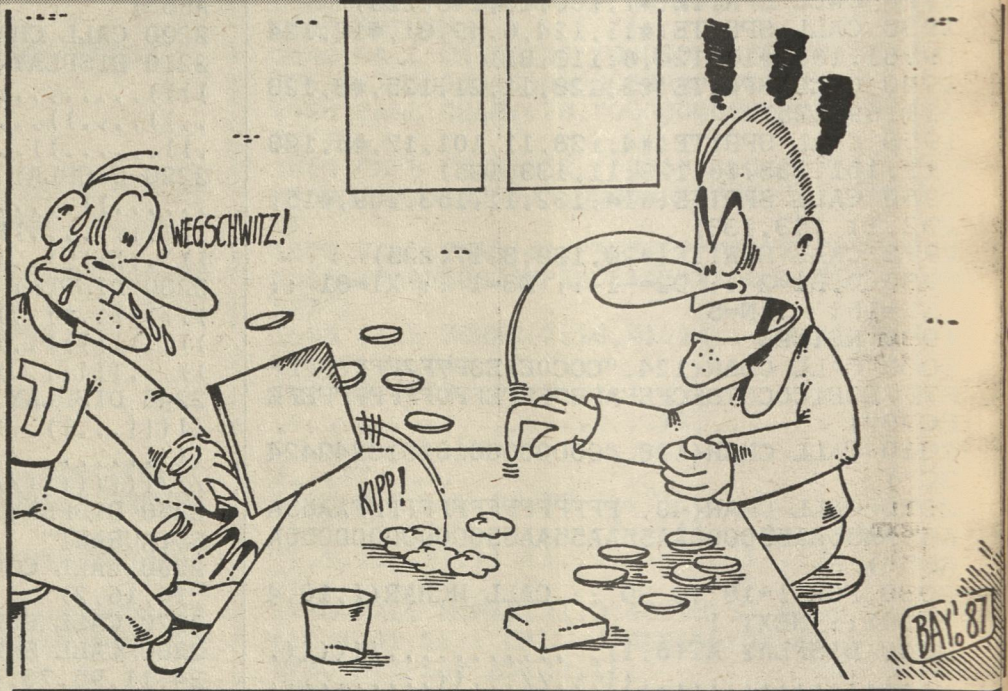
Vier gewinnt auf vier Ebenen!

Tetrade

Schalten Sie Ihr Gehirn auf Hochtouren! Volle Konzentration, denn: Strategie ist gefragt!

Als Gesellschaftsspiel sicherlich den meisten schon bekannt, haben wir das reizvolle Spiel „Vier gewinnt" für den Apple II. Der Spielverlauf ist schnell erklärt. Auf Ihrem Bildschirm baut sich ein Spieltürmchen mit vier Ebenen auf. Jede dieser Flächen besteht aus 4x4 Kästchen. Nachdem Sie sich zu Beginn entschieden haben, welches Ihr Spielsymbol ist (X oder O), kann das Duell mit dem Computer seinen Lauf nehmen. Wer als erster vier Symbole in eine Linie gebracht hat, der hat das Match gewonnen. Dabei müssen Sie darauf achten, daß die Linie über alle vier Ebenen laufen kann, und das diagonal, horizontal und vertikal.

Also Augen auf, sonst verlieren Sie blitzschnell den Überblick. Starten sollten Sie zunächst im ersten von drei Leveln. Glauben Sie uns ruhig, wenn wir Ihnen sagen, daß es dies schon verdammt in sich hat. Ran an die Buletten, aber mit Köpfchen.



„Jetzt reicht's! Immer wenn ich auf der Siegesstraße bin passiert so etwas!"

Das Listing

```
1 A = 20450: POKE A + 2,173: POKE
    A + 3,48: POKE A + 4,192: POKE
    A + 5,136: POKE A + 6,208: POKE
    A + 7,5: POKE A + 8,206: POKE
    A + 9,227: POKE A + 10,79: POKE
    A + 11,240: POKE A + 12,9: POKE
    A + 13,202: POKE A + 14,208:
     POKE A + 15,245: POKE A + 1
    6,174: POKE A + 17,226
2  POKE A + 18,79: POKE A + 19,7
    6: POKE A + 20,228: POKE A +
    21,79: POKE A + 22,96
3  HIMEM: 20449: LOMEM: 16385: GOTO
    5
4  POKE 20450,P: POKE 20451,D: CALL
    20452: RETURN
5  HOME
10  DIM A%(4,4,4),B%(76,5),C%(14
    0):SO = 1:AA = 1:NN = 2:NY =
    50:YY = 2: POKE  - 16368,0: GOSUB
    990: GOSUB 450: VTAB 20: CALL
     - 958
13  HOME : VTAB 10: HTAB 10: PRINT
    "WHICH LEVEL (1-3) ? ";: GET
    LE$:LE =  VAL (LE$): IF LE <
    1 OR LE > 3 THEN  CALL  - 95
    8: GOTO 13
14  PRINT LE$;
15  FOR G = 1 TO 100: NEXT G: HOME
    : VTAB 10: HTAB 10: PRINT "W
    HO GOES FIRST, X OR O ? ";: GET
    LE$: PRINT LE$;: FOR G = 1 TO
    100: NEXT G: IF LE$ = "X" THEN
     HOME : GOSUB 405: GOTO 25
16  IF LE$ = "O" THEN  HOME : GOTO
    19
17  VTAB 23: CALL  - 958: GOTO 1
    5
19  GOSUB 405: VTAB 21: PRINT "M
    Y MOVE  ": GOTO 200
25  HOME : VTAB 21: PRINT "YOUR
    MOVE"
30 XM = 1:YM = 1:LV = 1:GG = 1: POKE
     - 16368,0:X1 = X1 - 1:Y1 =
    Y1 - 2
35  IF A%(XM,YM,LV) > 0 THEN  ON
    GG GOTO 70,90
36  POKE  - 16368,0
40  HCOLOR= 3: GOSUB 86
41  HCOLOR= 0: GOSUB 160:X1 = AP
    :Y1 = BP: GOSUB 86:X1 = AP:Y
    1 = BP
43 PQ =  PEEK ( - 16336)
45 WW =  PEEK ( - 16384): IF WW =
    149 THEN QQ = 1: GOTO 70
50  IF WW = 136 THEN QQ =  - 1: GOTO
    90
55  IF WW = 141 THEN  HCOLOR= 3:
     GOSUB 86: GOTO 140
60  IF WW = 160 THEN 120
65  GOTO 40
70 XM = XM + 1: IF XM = 5 THEN X
    M = 1:YM = YM + 1
75  IF YM = 5 THEN YM = 1:LV = L
    V + 1
80  IF LV = 5 THEN LV = 1
83 GG = 1: GOTO 35
86  HPLOT 102 + X1,6 + Y1 TO 104
     + X1,6 + Y1: HPLOT 103 + X1
    ,5 + Y1 TO 103 + X1,7 + Y1: RETURN
90 XM = XM - 1: IF XM = 0 THEN Y
    M = YM - 1:XM = 4
100  IF YM = 0 THEN LV = LV - 1:
    YM = 4
110  IF LV = 0 THEN LV = 4
113 GG = 2: GOTO 35
120 LV = LV + 1: IF LV = 5 THEN
    LV = 1
125 XM = 1:YM = 1:GG = 1: GOTO 3
    5
140 ZX = XM:ZY = YM:ZL = LV:A%(X
    M,YM,LV) = 1:FW = 1: GOSUB 6
```

```
100 !**********************      *                    *      *    INDIANA JONES   *      *   ---------------  *      *
*
110 !*  THOMAS DREIBRODT  *      *                    *      *   OTTILIENSTR. 4   *      *                    *      *   4630 BOCHUM
7   *
120 !*                    *      * TEL.:(0234) 287222 *      *                    *      *   (C) BY TD.1985   *      *
*
130 !*                    *      * EXT.BASIC+JOYSTICK *      *                    *      **********************
140 CALL SCREEN(2) :: CALL CLEAR :: FOR I=1 TO 13 :: CALL COLOR(I,7,1) :: NEXT I
150 DATA FEFEFE3838383838,C6C6C6E6E6F6F6DE,F1F9CCCCCCCCCCCC,E3E2C6C6C6CCCFCC,A4A4B4B4BDADADA5A5,60A0A0A020E02020
160 DATA 3838383838FEFEFE,CECEC6C6C6C6C,CCCCCDF9F,CCCCECE,A5A48
170 DATA FFFFFF0707070707,1E3F336161616161,333333BBBBBFBFB7,7C7D616179786060,F8F88080F0F81818,0707C7C7C7FEFE7C
180 DATA 616161333F1E,B7B3B333,617D7C,F8F
190 DATA 071E1D3E3D3A3D3A,FC83008000810181,00C0201010080808,0000000000071F3E,7D7A757A75FE5FBF,448241A040A9FFFF
200 DATA 081838387FF0C080,00000000E01C0202,7DFAFDFA7D1F0F00,53A855AA55AAFD57,FE7000A050A855AA,0000000003040C94,040810608
210 DATA 68536D342A355A55,D5BF820282539740,64F46C0408080A0B,077D2E755A3E5D6B,EA75FA7DBD7EBF1F,BFC09F40A8D47F80,D717972F5FBF3F1F
220 DATA FEA5A4929280A424,542D361F10090000,DFCFEF47D79360B4,C0C0C0C0C0C0E020,1F1E1C1811111012,00494900202240440
230 DATA 0000000001070A14,00000F74A292547F,0000C038A49725C8,2A316955A2D4ACC4,60C0800000000000,310E020100000000
240 DATA 010101010103020 3,A4548C44C4289808,020506C536050604,C83090109C139010,000000000000E010,7FAAD57F2A556A3F,F90001FE00010EF0
250 DATA 488844A444A444A4,556A3F1A0F0006,0102FC0157FC0060,48F0A06080000000,050A0D1A150A0D07,10C8281C048848F0
255 !**********************     *Coming soon from IJS:*     *                    *     * RETURN OF THE JEDI  *
**********************
260 DATA 000000000000E090,981412E6A65A4933,1D151B1519171915
270 DATA 000001070A0D1A15,0000010307060D0E,366CF0A850A850A8,0000000000000003,00000001010107F9
280 DATA 1520DBB924B624B6,540860D894DA9FD0,0000000000F80402,00000000000000FE,060D1A141B152D15,5101788432424232,24B6DB0040404272
290 DATA 90E0003F08085E49,010000F800000B0A,010000000000BBAA,00C0201010109010,1A152A554A3F0000,847880A4AAD67C00,49487000CC2A1101
300 DATA C9899E0840A89502,0B084B0000F10A04,BBA8BB0000F60901,8888883020606080
310 DATA F34949F3414141C3,CF2424CF8444242F,394A423108084A73,C245058547484888,25353434ACACADA5,2622A2A3622222C6
320 DATA 5E88889E8848485E,9292525232129062
330 !
340 FOR I=33 TO 132 :: READ A$ :: CALL CHAR(I,A$) :: NEXT I
350 DISPLAY AT(1,10):"!""#$%&,-../0": TAB (10);"'()*+ 12345"
360 DISPLAY AT(8,1):"            678  PQRc                 9:;<= STUd                  >?@ABVW  e                   CDE XYZ e"
370 DISPLAY AT(12,1):"          fFGHIJ[\] e              ghKLMNO^_` e              ijklmn   ab  e              opqrstu"
380 DISPLAY AT(16,1):"       vwxyz{|                 "
390 DISPLAY AT(21,11):"}~......"
400 CALL MUSIK
410 CALL KEY(0,N,S) :: IF S=0 THEN 410 ELSE CALL CLEAR :: CALL CHARSET
420 FOR I=0 TO 13 :: CALL COLOR(I,7,1) :: NEXT I :: CALL COLOR(14,6,1)
430 DATA 1C3E33637F7F6363,7E7F333E33337F7E,1F3F606060603F1F,7C7E333333337E7C,7F7F307E7E307F7F,7F7F307E7E303030,1E3F606767633F1E
440 DATA 6363637F7F636363,7F7F181818187F7F,7F7F030303637E3C,63666C78786C6663,6060606060607F7F,637F7F6B6B6B6363,63737B7F6F676363
450 DATA 3E7F636363637F3E,7E7F337F7E303030,3E7F63636B677F3E,7E7F337F7E3C3633,3F7F607E3F037F7E,7F7F181818181818,6363636363637F3E
460 DATA 6363633636361C1C,63636B6B3E3E3E,63637F1C1C7F6363,63633F3F060C1818,7F7F060C18307F7F
470 DATA E79494E7868584,3CA1A13C2020BD,EF0808CF2828CF,4B496979594949,DE10101E10101E,E19191919191E1,C92929C72121C1,00
480 DATA FEFE383838383838,FEFE0E0E0E0E0E0E,3F7FE0E0E0E0E07E,383838393938FEFE,0E0E0ECECEFEFE7C,7F0707070707FFFE
490 CALL CHAR(60,"101010FE101010",61,"0010107C1010",62,"000010381")
500 !
510 FOR I=97 TO 122 :: READ A$ :: CALL CHAR(I,A$) :: NEXT I
520 FOR I=128 TO 141 :: READ A$ :: CALL CHAR(I,A$) :: NEXT I
530 DISPLAY AT(1,9):"indiana jones" :: DISPLAY AT(3,9):"         ....... ...                          ..."
540 DISPLAY AT(8,1):"(C) BY THOMAS DREIBRODT 1985                                 OTTILIENSTR..4 "
550 DISPLAY AT(12,1):"        4630 BOCHUM-7                                      TEL:(0234)/287222 "
560 DISPLAY AT(22,3):"   READY-PRESS ANY KEY"
570 !
580 CALL SPRITE(#1,60,16,22,150,#2,62,16,36,174)
590 FOR I=1 TO 3 :: CALL PATTERN(#1,59+I,#2,63-I) :: FOR J=1 TO 4 :: CALL KEY(0,N,S) :: IF S THEN 610
600 NEXT J :: NEXT I :: GOTO 590
610 CALL CLEAR :: CALL DELSPRITE(ALL)
620 DATA "marshall college 1936.....","der archaeologe indiana","jones entdeckt durch zufall","in einer alten aztekenstatue"
630 DATA "eine alte landkarte,in der","der standort des legendaeren","sonnentempels eingezeichnet","ist..einer alten aztekensage"
640 DATA "nach wurde hier'das auge","quetzalcoatls',des schlangen","gottes,aufbewahrt,ein ","riesiger diamant,der,der "
650 DATA "sage nach,in den haenden des","richtigen mannes zu neuen","leben erwachen wird.....","","""
660 DATA "suedamerika 1936","nach langen rekonstruktionen","konnte indy den standort des","sonnentempels ermitteln:"
670 DATA "mitten im tiefsten dschungel","von peru..","","nach langen wanderungen im","urwald und nach einigen aus-"
680 DATA "einandersetzungen mit indios","und deren priester quontec-","mar hat indy endlich sein","ziel vor augen:"
690 DATA "den sonnentempel und das","'auge des quetzalcoatls'.....","doch vorsicht ist geboten","denn wer weiss,wer oder was"
700 DATA "'das auge des quetzalcoatls'","bewacht.....","","","","","","","","","
710 FOR I=1 TO 44 :: READ A$ :: PRINT A$ :: FOR J=1 TO 2 :: PRINT :: FOR K=1 TO 140 :: NEXT K :: NEXT J :: NEXT I
720 CALL CHAR(96,"81C56F3FBFFFFFFF",97,"40202AB877BF7FFF",98,"01CAE4D5767FEFFF",99,"00080A505E7FFFFF")
730 CALL CHAR(100,"DF7776BBAFBACFDE",101,"000280182C5C3006",102,"00E0420C10280220",103,"0042100040042B04")
740 CALL CHAR(104,"6E566E302C342C34",105,"2C566E342C342C34",106,"2C342C342C526E3C")
750 CALL CHAR(107,"D7AFD7AF706E566E",108,"56AFD76E566E566E",109,"566E566EB1DFAF7E")
760 CALL CHAR(110,"D450400FFBDBDFF0",111,"000FFD6F6FF0",112,"8020A050A854AA55",113,"AA55AA55AA55AA55")
770 CALL CHAR(40,"FFFFFFF76545",41,"FCF4D4D0D080",42,"FFFFFFDFDF4901",43,"2E322A322A362A32",44,"1F2F5F6F57ABD5EA")
780 CALL CHAR(45,"C0FCFEFFFFFF7FBF",46,"D5AA2A151A1F0F07",47,"55AB55AB55AA5CF8")
790 CALL CHAR(49,"000C7FEFFBBFFFFFF",50,"010307173FBFFFFF",51,"82A6FFFFFFFFFFFF",52,"80E8F8FEFEFFFFFF")
800 CALL CHAR(56,"0305060D0A3D66FF",57,"008040A0F0F0F8FC",58,"000101030F3F7FFF",59,"FFFFFFFFFFFFFFFF",60,"80C0E0FEFFFFFFFF")
810 CALL CHAR(61,"0000002070F8FCFF",62,"0000000000000000")
820 CALL COLOR(9,13,10,10,15,2,2,4,13,3,13,4,4,4,8)
830 DISPLAY
AT(1,1):">>>>
```

```
1060 CALL MOTION(#1,A,5,#2,A,5) :: CALL PATTERN(#2,136) :: FOR I=1 TO 40 :: NEXT I :: CALL PATTERN(#2,132) :: FOR I=1 TO 40 :: NEXT I
1070 CALL MOTION(#1,0,0,#2,0,0) :: RETURN !GEHEN
1080 CALL POSITION(#1,Y,X) :: IF X>235 THEN 980
1090 CALL SPRITE(#3,116,2,Y,X+16) :: CALL PATTERN(#1,120) :: FOR I=1 TO 100 :: NEXT I :: CALL DELSPRITE(#3) :: CALL
PATTERN(#1,128) :: RETURN
1100 SUB MUSIK
1110 CALL SOUND(350,262,5,262,10) :: CALL SOUND(125,277,5,277,10) :: CALL SOUND(125,311,2,311,10)
1120 CALL SOUND(400,415,0,415,0,415,5) :: FOR I=1 TO 50 :: NEXT I :: CALL SOUND(200,262,5,262,10) :: CALL SOUND(125,277,3,277,10)
1130 FOR I=1 TO 30 :: CALL SOUND(-500,311,I,311,I) :: FOR J=1 TO 10 :: NEXT J :: NEXT I
1140 SUBEND
```

---------- TEIL 1 (name file indiana1)

```
100 !************************     *    INDIANA JONES    *     *HAUPTPROGRAMM (TEIL1)*     *(C)  THOMAS DREIBRODT*
************************
110 CALL MAGNIFY(3) :: CALL CLEAR :: CALL SCREEN(2) :: RANDOMIZE :: CALL COLOR(1,13,1,2,3,1,5,3,4,3,13,1,4,13,1,13,7,1,7,16,1)
120 CALL COLOR(8,16,1,6,13,1,14,7,1,12,7,1,11,7,1)
130 CALL CHARPAT(63,A$) :: IF A$="0038440408100010" THEN PRINT :"GRAFIKDATEN NICHT VORHANDEN": :"BITTE TITELPROGRAMM LADEN" :: BREAK
140 CALL CHAR(33,"FFBF5F2E2E2E2616",34,"1616160E0E0C0404",35,"FFFF6F676642",36,"FFF66A64F4E06040",37,"20041202200A4012")
150 CALL CHAR(38,"2200082200441084",39,"386EFFF55FFFF67C",40,"00001092DADAFFFF",41,"000004044656D7FF",42,"000044446C6EFFFF")
160 CALL CHAR(43,"0002000801042002",46,"C0F0EEBFFB7FBEFC",45,"073947BDBF5F271F",65,"000008585A5EFFFF",66,"7C82CDCD11565454")
170 CALL CHAR(67,"1028124500084400",68,"0000024243EBFBFF",69,"071D377EDFFB7F1F",70,"F0B8ECDE7EFA7CF0",71,"3C7EDFFBDFB73E")
180 CALL CHAR(58,"3844404039054538",59,"000000E7080808E7",60,"0000002C92909010",61,"00006290E08072")
190 CALL CHAR(72,"F8444444454545F8",73,"000000E6252525F5",74,"0000008E5252524F",75,"000000718A7B0A71",76,"00000088408000C8")
200 CALL CHAR(92,"001008221B1F0FCF0F1F3860040919300018102004B0E0C08F80D04804068400")
210 CALL CHAR(84,"0003070507030301 0F112F213E3036310080C040C0808000E010E808F818D818")
220 CALL CHAR(88,"676F676EECAC2C0C0C0E0C0606060E3ECCECCCE6676A6A60607030303030383E")
230 BILD=5 :: DAMAGE=0 :: WAVE=1 :: CALL CHARPAT(63,A$) :: SCORE=VAL(A$)
240 CALL FIGUREN(WAVE)
250 DISPLAY
AT(1,1):"'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..'-.."
260 DISPLAY AT(4,1):"#!$#$#$$$$!#!!!$$##$!$$!#!$$ ""        "" """""     ""  "" "" "
270 DISPLAY AT(6,1):"   %   &   & &  %  %  &   %    &   %   % %  &  &  %&  &   &    +%  +%&  %  +%  %  +&   %   %    &&   &  &  &
%"
280 DISPLAY AT(10,1):"*)()*()()*)*)*)()()***)*)*CCCGCCCCCCCGCCEFCCBCCCCCCCCCCCCCCEFCCCCCCCCCCCCCCCCCCCCCC"
290 IF WAVE=2 THEN GOSUB 1330
300 DISPLAY
AT(13,1):"GADADADGADADAGADADAGGADADAADEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEEFEFEFEFEFEFEFEFEFEFEFEFEF"
310 DISPLAY AT(18,1):"'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..                        -..-.. :;<=        xzt ..  .. -..-..
y{u ..  .. -.."
320 DISPLAY AT(22,1):"-.. HIJKL       ... ..  |~ -..-..             ..  ..  }. -..-..'-..'-..'-..'-..'-..'-..'-..'-..'-"
330 CALL MUSIK2
340 FI,SN,TR1,TR2=0
350 GOSUB 1400 :: CALL SPRITE(#10,128,7,60,40,#11,132,7,76,40)
360 ON WAVE GOSUB 1150,1190
370 CALL JOYST(1,A,B) :: CALL KEY(1,N,S) :: IF S THEN 1230
380 IF B=4 THEN 1270 ELSE IF A THEN GOSUB 1310
390 IF WAVE=2 THEN 420
400 IF SN THEN CALL PATTERN(#5,108)
410 IF FI THEN CALL PATTERN(#7,96)
420 IF WAVE=1 THEN 460 ELSE IF TR1 AND INT(RND*4+1)=4 THEN CALL LOCATE(#20,200,X1)
430 IF TR1 AND INT(RND*4+1)=1 THEN CALL LOCATE(#20,80,X1)
440 IF TR2 AND INT(RND*4+1)=4 THEN CALL LOCATE(#7,47,145) :: CALL SOUND(10,-5,5)
450 IF TR2 AND INT(RND*4+1)=1 THEN CALL LOCATE(#7,36,145)
460 CALL COINC(#11,#5,15,C) :: IF C THEN 990 ELSE CALL COINC(#11,#20,8,C) :: IF C THEN 990 ELSE CALL COINC(#10,#7,15,C) :: IF C THEN
990
470 IF SN THEN CALL PATTERN(#5,112)
480 IF FI THEN CALL PATTERN(#7,104)
490 CALL POSITION(#10,Y,X) :: CALL LOCATE(#10,Y,X,#11,Y+16,X) :: IF X>220 THEN 510 ELSE IF X<40 THEN CALL LOCATE(#10,60,40,#11,76,40)
500 GOTO 370
510 CALL DELSPRITE(ALL) :: SCORE=SCORE+100 :: BILD=BILD+1 :: IF BILD>=10 THEN 520 ELSE 340
520 CALL MUSIK2 :: BILD=5 :: WAVE=WAVE+1 :: CALL DELSPRITE(ALL) :: IF WAVE<>3 THEN 240
530 WA=5 :: DEM=3 :: SCORE=SCORE+200 :: IF DEMAGE<5 THEN DEMAGE=DEMAGE-3
540 CALL FIGUREN(3) :: CALL CHAR(77,"6B576B3F1B171B17",78,"FEFEFE04F8F838F8",79,"1B151B151B151B15",80,"F838F8C8F898F8F8")
550 CALL CHAR(81,"1B151B151B2F556B",82,"C8F898F0E8FCFEFE") :: DISPLAY AT(4,1):"##$#MN##$#MN#$$$MN$$#$MN$#$# %& OP%&  OP   %OP & &OP &
%&OP %% OP &&&OP  %%OP &  "
560 DISPLAY AT(7,1):"  %%OP  &&OP %& OP  %%OP &&   & OP%%% OP  %%OP  % OP  %% %% OP&&  OP && OP %& OP  &&*()(QR*()(QR(()*QR()(*QR*()
("
570 GOSUB 1400 :: CALL SPRITE(#10,128,7,50,30,#11,132,7,66,30)
580 CALL SPRITE(#20,92,10,50,220,#21,96,9,66,220)
590 CALL JOYST(1,A,B) :: IF A THEN GOSUB 1310
600 CALL KEY(1,N,S) :: IF S THEN 860
610 CALL POSITION(#10,Y,X) :: CALL LOCATE(#11,Y+16,X)
620 IF INT(RND*2+1)=1 THEN GOSUB 800
630 IF INT(RND*3+1)=3 THEN GOSUB 830
640 GOTO 590
650 FOR I=1 TO 10 :: CALL SOUND(50,330,5,220,10) :: SCORE=SCORE+100 :: GOSUB 1400 :: NEXT I
660 FOR I=DAMAGE TO ..5 STEP -..5 :: CALL COLOR(#10,16,#11,16) :: CALL SOUND(-500,I*400+110,5,I*500+110,10,110,15) :: CALL
COLOR(#10,7,#11,7)
670 DAMAGE=DAMAGE-..5 :: GOSUB 1400 :: NEXT I :: CALL MUSIK1 :: FOR I=1 TO 400 :: NEXT I
680 CALL DELSPRITE(ALL) :: CALL CLEAR :: CALL CHARSET :: FOR I=0 TO 14 :: CALL COLOR(I,7,1) :: NEXT I
690 SC$=STR$(SCORE) :: L=LEN(SC$) :: CALL CHAR(63,RPT$("0",16-L)&SC$)
700 DISPLAY AT(1,2):"INDIANA JONES" :: DISPLAY AT(3,1):" (C) BY THOMAS DREIBRODT" :: DISPLAY AT(7,4):"BITTE WAEHLEN:"
710 DISPLAY AT(10,4):"NOCHMAL SPIEL 1" :: DISPLAY AT(13,4):"SPIEL 2 NACHLADEN"
720 M=13
730 CALL HCHAR(M,4,42)
740 CALL JOYST(1,A,B) :: CALL KEY(1,N,S) :: IF S THEN 770
750 CALL VCHAR(5,4,32,14) :: IF B=4 THEN M=10 ELSE IF B=-4 THEN M=13
760 GOTO 730
770 IF M=10 THEN CALL CLEAR :: RUN
780 CALL CLEAR :: PRINT "NACHLADEN,BITTE WARTEN.....": : :"* LOADING I..JONES (PART 2)": :"* DON'T"
790 RUN "DSK1..INDIANA2"
800 CALL MOTION(#21,0,-5,#20,0,-5) :: CALL PATTERN(#21,104) :: CALL SOUND(10,-7,5,110,10) :: FOR J=1 TO 20 :: NEXT J :: CALL
PATTERN(#21,96)
810 CALL SOUND(10,-7,5,220,10) :: FOR J=1 TO 20 :: NEXT J :: CALL MOTION(#20,0,0,#21,0,0) :: CALL POSITION(#20,Y2,X2)
820 CALL LOCATE(#20,Y2,X2,#21,Y2+16,X2) :: RETURN
830 CALL POSITION(#20,Y2,X2) :: IF X2<30 THEN RETURN ELSE CALL PATTERN(#20,100) :: CALL SPRITE(#22,108,10,Y2,X2-16)
840 CALL SOUND(10,-5,5,110,10) :: FOR J=1 TO 20 :: NEXT J :: CALL COINC(#22,#10,5,C) :: CALL DELSPRITE(#22) :: CALL
PATTERN(#20,92) :: IF C THEN 990
850 RETURN
860 CALL POSITION(#10,Y,X) :: IF X>230 THEN 620 ELSE CALL PATTERN(#10,120) :: CALL SPRITE(#12,116,7,Y,X+16) :: CALL SOUND(10,-5,3)
```

```
870 CALL COINC(#12,#20,5,C) :: IF C THEN 890
880 FOR I=1 TO 40 :: NEXT I :: CALL DELSPRITE(#12) :: CALL PATTERN(#10,128) :: GOTO 620
890 IF DEM=0 THEN 920 ELSE CALL MOTION(#20,2,19,#21,2,19) :: CALL SOUND(10,-7,5) :: SCORE=SCORE+50
900 CALL SOUND(-100,-5,15) :: CALL MOTION(#20,0,0,#21,0,0) :: CALL POSITION(#20,Y2,X2) :: CALL LOCATE(#20,50,X2,#21,66,X2)
910 GOSUB 1400 :: CALL DELSPRITE(#12) :: CALL PATTERN(#10,128) :: DEM=DEM-1 :: GOTO 620
920 CALL MOTION(#20,3,45,#21,3,45) :: CALL SOUND(-100,-7,15) :: FOR I=5000 TO 1500 STEP -500 :: CALL SOUND(-500,I,0,I,10,110,20) ::
NEXT I
930 SCORE=SCORE+100 :: DEM=3 :: WA=WA-1
940 CALL DELSPRITE(#20,#21,#12) :: CALL PATTERN(#10,128) :: GOSUB 1400 :: IF WA=0 THEN 650
950 CALL SPRITE(#20,92,10,50,230,#21,96,7,66,230) :: GOTO 590
960 CALL MOTION(#6,0,0,#7,0,0) :: CALL POSITION(#7,Y1,X1) :: CALL MOTION(#6,80,0,#7,-80,0) :: FOR I=1 TO 7 STEP 3
970 CALL SOUND(-500,110,30,110,30,110+2^(8-I)*100,30,-8,6) :: NEXT I :: CALL DELSPRITE(#6,#7)
980 SCORE=SCORE+50 :: GOSUB 1400 :: CALL SPRITE(#6,100,11,55,180,#7,104,7,55,180) :: CALL MOTION(#6,0,-12,#7,0,-12) :: RETURN
990 CALL MOTION(#10,0,0,#11,0,0) :: CALL POSITION(#10,Y,X) :: IF WAVE=3 THEN 1000 ELSE CALL SPRITE(#9,92,9,Y+5,X)
1000 CALL SOUND(10,-5,0) :: FOR I=1 TO 20 :: NEXT I :: CALL DELSPRITE(#9)
1010 CALL PATTERN(#11,124) :: CALL LOCATE(#10,Y+10,X,#11,Y+21,X) :: FOR I=1 TO 100 :: NEXT I
1020 DAMAGE=DAMAGE+..5 :: GOSUB 1400 :: IF DAMAGE>=10 THEN 1030 ELSE CALL DELSPRITE(ALL) :: IF WAVE=3 THEN 570 ELSE 350
1030 CALL DELSPRITE(#5,#6,#7,#20,#21) :: CALL PATTERN(#11,132) :: CALL LOCATE(#10,Y,X,#11,Y+16,X)
1040 FOR I=1 TO 50 :: NEXT I :: CALL SPRITE(#8,92,9,Y,X-2,#9,92,9,Y,X+2)
1050 FOR I=3500 TO 500 STEP -500 :: CALL SCREEN(2) :: CALL SPRITE(#I/500,92,9,INT(RND*10+8)+Y,INT(RND*-3+6)+X)
1060 CALL SOUND(-4000,110,30,110,30,I,30,-8,0) :: CALL SCREEN(16) :: NEXT I
1070 CALL SCREEN(2)
1080 CALL PATTERN(#10,84,#11,88) :: CALL COLOR(#10,16,#11,16) :: CALL SOUND(-250,-5,0) :: CALL MOTION(#8,0,-80,#9,0,80) :: FOR I=1 TO
50 :: NEXT I
1090 CALL DELSPRITE(#8,#9)
1100 FOR I=1 TO 7 :: CALL MOTION(#I,3,INT(RND*-1+2)) :: NEXT I :: FOR I=1 TO 120 :: NEXT I :: FOR I=1 TO 7 :: CALL DELSPRITE(#I) ::
NEXT I
1110 FOR I=1 TO 200 :: NEXT I :: CALL MOTION(#10,55,17) :: FOR I=1 TO 10 :: NEXT I :: CALL MOTION(#10,0,0)
1120 CALL SOUND(-250,-7,0) :: FOR I=1 TO 30 STEP 3 :: CALL SOUND(-500,-5,I) :: NEXT I
1130 CALL MUSIK1
1140 CALL KEY(1,N,S) :: IF S THEN CALL DELSPRITE(ALL) :: RUN ELSE 1140
1150 IF INT(RND*2+1)=1 THEN CALL SPRITE(#20,96,2,80,INT(RND*170+50))
1160 IF INT(RND*2+1)=2 THEN CALL SPRITE(#5,112,7,75,70,0,INT(RND*10+15)) :: SN=1
1170 IF INT(RND*2+1)=2 THEN CALL SPRITE(#7,104,9,55,230,#6,100,11,55,230) :: CALL MOTION(#7,0,-12,#6,0,-12)
1180 FI=1 :: RETURN
1190 IF INT(RND*2+1)=2 THEN X1=INT(RND*180+45) :: CALL SPRITE(#20,96,2,80,X1) :: TR1=1
1200 IF INT(RND*2+1)=2 THEN CALL SPRITE(#5,112,16,75,230,0,-15)
1210 CALL SPRITE(#21,100,8,25,145,#6,100,8,36,145,#7,104,8,36,145) :: TR2=1
1220 RETURN
1230 CALL POSITION(#10,Y,X) :: IF X>230 THEN 370 ELSE CALL PATTERN(#10,120) :: CALL SPRITE(#12,116,7,Y,X+16) :: CALL SOUND(-10,-5,3)
1240 CALL COINC(#12,#7,10,C) :: CALL COINC(#10,#7,10,C1) :: IF C AND WAVE=1 THEN GOSUB 960 ELSE IF C1 THEN 990
1250 FOR I=1 TO 40 :: NEXT I :: CALL DELSPRITE(#12) :: CALL PATTERN(#10,128)
1260 GOTO 400
1270 CALL PATTERN(#10,140,#11,124) :: FOR I=-12 TO 12 STEP 2 :: CALL MOTION(#10,I,A,#11,I,A)
1280 CALL COINC(#11,#5,10,C) :: CALL COINC(#10,#7,15,C1) :: IF C OR C1 THEN CALL MOTION(#10,0,0,#11,0,0) :: GOTO 990
1290 CALL SOUND(-400,-5,20) :: NEXT I :: CALL MOTION(#10,0,0,#11,0,0) :: CALL POSITION(#11,Y,X) ::
1300 CALL LOCATE(#10,Y-16,X) :: CALL PATTERN(#10,128,#11,132) :: SCORE=SCORE+10 :: GOSUB 1400 :: GOTO 400
1310 CALL MOTION(#10,0,A*2..5,#11,0,A*2..5) :: CALL PATTERN(#11,136) :: CALL SOUND(-15,-6,10) :: FOR I=1 TO 30 :: NEXT I
1320 CALL PATTERN(#11,132) :: CALL SOUND(-15,-6,10) :: FOR I=1 TO 30 :: NEXT I :: CALL MOTION(#10,0,0,#11,0,0) :: RETURN
1330 CALL CHAR(77,"00012A3F203F70EA",78,"0080FCD41CE496AB",79,"FEFDFEBFBF990E07",80,"87C7478585E910E0",81,"352A352AA7FD5AA7F")
1340 CALL COLOR(6,15,1,7,15,1)
1350 CALL CHAR(82,"FCFCFCFC02FFFFF7") :: FOR I=1 TO 3 :: DISPLAY AT(8,I*7) SIZE(2):"MN" :: DISPLAY AT(9,I*7) SIZE(2):"OP"
1360 DISPLAY AT(10,I*7) SIZE(2):"QR" :: NEXT I
1370 DISPLAY AT(9,3) SIZE(2):"TV" :: DISPLAY AT(10,3) SIZE(2):"UW" :: DISPLAY AT(7,25) SIZE(2):"TV" :: DISPLAY AT(8,25) SIZE(2):"UW"
1380 DISPLAY AT(9,25) SIZE(2):"XZ" :: DISPLAY AT(10,25) SIZE(2):"Y["
1390 GOTO 300
1400 DISPLAY AT(20,9) SIZE(7):SCORE :: DISPLAY AT(22,9) SIZE(4):DAMAGE*10 :: RETURN
1410 SUB FIGUREN(E)
1420 ON E GOTO 1430,1490,1540
1430 CALL CHAR(112,"0000404040603030303067 6FCCD8F8701C373C3A381818180C0C8686CCCCF870")
1440 CALL CHAR(108,"4030181818306060C0C0C06033371E0C0020597E79C0C06030180C0683C77E3C")
1450 CALL CHAR(104,"00071F1F1F3F7F7F3F1F3F7F3F13000000C0E0F0FEFEFCFCFEFCFCFCF8F0C00000")
1460 CALL CHAR(100,"0002100413174727071703092400000000C040D0EEF2FCFCF2F4E48830C00000")
1470 CALL CHAR(96,"000000010F1F3F7F7FFF7F3F1F0F010000000000DEFFFFFFFFFEFFFEFCF0A000")
1480 SUBEXIT
1490 CALL CHAR(100,"DBFBFFDFDBDBDBDBDBDBDB9B130200006C6C6CECFF7F6F6C6C6C6C6C6C6C4C08")
1500 CALL CHAR(104,"DBDBDBDBDBDBDBDBDBDBDB9B130200006C6C6C6C6C6C6C6C6C6C6C6C4C08")
1510 CALL CHAR(96,"00000000FFFFFF7F7F3F3F1F1F1F000000000000F8F8F8FCFCFEFEFFFFFF0000")
1520 CALL CHAR(112,"00000000000040FF40000000000000000000000000000A0C0A000000000000000")
1530 SUBEXIT
1540 CALL CHAR(92,"0103030103014170703D1F0F0F070301E0F8FCC870F0C060B0787878F4ECD8B8")
1550 CALL CHAR(96,"03030707070E0E0E0E1F1E0E0E0C3C7DF0F8F8F8F8FCFCFE7E7E7E3A3830F0F0")
1560 CALL CHAR(100,"070F0E070D0707FFFFFFFF080F0F1F1FC0E070E0C0C000C0E0E0D030F0F0E0F0")
1570 CALL CHAR(104,"1F1F0F1F3F3F7D7C7C7C7C1C181878F8F0E0F0F0F8FCFEFFFFFFFC7E06061E3E")
1580 CALL CHAR(108,RPT$("0",44)&"207F7F77070604")
1590 SUBEND
1600 SUB MUSIK1
1610 FOR I=1 TO 10 :: CALL SOUND(-500,247,I,494,I,123,I*3) :: NEXT I :: CALL SOUND(250,262,3) :: CALL SOUND(250,294,1,584,10)
1620 CALL SOUND(300,330,0,330,0,659,5) :: CALL SOUND(250,294,2,587,10) :: CALL SOUND(200,262,3,523,10) :: FOR I=1 TO 15
1630 CALL SOUND(-500,330,I,330,I+5,659,I*2) :: NEXT I :: CALL SOUND(200,330,4,659,4) :: CALL SOUND(200,349,4,698,4) ::
1640 CALL SOUND(200,330,4,659,4) :: CALL SOUND(200,294,3,587,3) :: CALL SOUND(200,262,3,523,3)
1650 CALL SOUND(100,294,4,587,6) :: CALL SOUND(100,262,4,523,6) :: FOR I=1 TO 20 :: CALL SOUND(-500,247,I,494,I+4,988,I+10) :: NEXT I
1660 SUBEND
1670 SUB MUSIK2
1680 CALL SOUND(400,262,2) :: CALL SOUND(200,277,3,554,8) :: CALL SOUND(200,311,2,622,5) :: CALL SOUND(400,415,0,415,0,831,3)
1690 CALL SOUND(200,262,2,523,5) :: CALL SOUND(200,277,2,554,5) :: FOR I=1 TO 10 :: CALL SOUND(-500,311,I,311,I*2,622,I*3) :: NEXT I
1700 SUBEND
```

---------- Teil 2 (name file indiana2)

```
100 !**********************     *    INDIANA JONES    *     *HAUPTPROGRAMM (TEIL2)*     *(C)  THOMAS DREIBRODT*
**********************
110 CALL MAGNIFY(3) :: CALL CLEAR :: CALL SCREEN(2) :: RANDOMIZE :: CALL COLOR(1,13,1,2,3,1,5,3,4,3,13,1,4,13,1,13,7,1,7,9,12)
120 CALL COLOR(8,16,1,6,9,1,14,7,1,12,7,1,11,7,1)
130 CALL CHARPAT(63,A$) :: IF A$="0038440408100010" THEN PRINT :"GRAFIKDATEN NICHT VORHANDEN": :"BITTE TITELPROGRAMM LADEN" :: BREAK
140 CALL CHAR(33,"FFBF5F2E2E2E2616",34,"1616160E0E0C0404",35,"FFFF6F676642",36,"FFF66A64F4E06040",37,"20041202200A4012")
150 CALL CHAR(38,"220008220044108 4",39,"386EFFF55FFFF67C",40,"00001092DADAFFFF",41,"000004044656D7FF",42,"000044446C6EFFFF")
155 CALL CHAR(79,"000070FFFFCFD6F9FF",78,"187CFFFFBBFAFFFF",80,"F3DDFF9F6FFDFFFF",81,"FFF1EEFF9EFF9F6F")
160 CALL CHAR(43,"0002000801042002",46,"C0F0EEBFFB7FBEFC",45,"073947BDBF5F271F",65,"000008585A5EFFFF",66,"7C82CDCD11565454")
170 CALL CHAR(67,"1028124500084400",68,"0000024243EBFBFF",69,"071D377EDFFB7F1F",70,"F0B8ECDE7EFA7CF0",71,"3C7EDFFBDFB73E")
180 CALL CHAR(58,"3844404039054538",59,"000000E7080808E7",60,"0000002C92909010",61,"00006290E08072")
190 CALL CHAR(72,"F8444444454545F8",73,"000000E6252525F5",74,"0000008E5252524F",75,"000000718A7B0A71",76,"00000088408000C8")
200 CALL CHAR(92,"001008221B1F0FCF0F1F3860040919300018102004B0E0C08F80D04804068400")
210 CALL CHAR(84,"0003070507030301 0F112F213E3036310080C040C0808000E010E808F818D818")
```

```
220 CALL CHAR(88,"676F676EECAC2C0C0C0E0C0606060E3ECCECCCE6676A6A60607030303030383E")
230 BILD=5 :: DAMAGE=0 :: WAVE=1 :: CALL CHARPAT(63,A$) :: SCORE=VAL(A$)
240 CALL FIGUREN(WAVE)
250 DISPLAY AT(1,1):RPT$("'-..",40)
260 DISPLAY AT(4,1):"#$$#$'-..-..$$#$#$#'-..-..#$$#$# &+  #$!## %& &  $#!#$ %& +  +% %  "" %&   &  & "" &%  %"
270 DISPLAY AT(7,1):"  &    +%  +%&  %  +%  %  +&   %   %    &&   &  &  &   %  &    +&  %&   %%  %% &&  &"
280 DISPLAY AT(10,1):"*)())NONON(*)*)*)OONON**)*)*CCCGBQPQPQCGCGCCGPQPQPCCGGCCCCCCGQPQPQCCGCCGGQQQPPCCCCCC"
300 DISPLAY
AT(13,1):"GADADQPQPPADAGADAPQPPQDADAADEFEFEQPPQQEFEFEFEQPPQPEFEFEFFEFEFPPQQPFEFEFEFPQQPPFEFEFEEFEFEQQPPPEFEFEFEQPPQQEFEFEF"
310 DISPLAY AT(18,1):"'-..'-..'-..'-..'-..'-..'-..'-..'-..'-..                        -..-.. :;<=        xzt ..  .. -..-..
y{u ..  .. -.."
320 DISPLAY AT(22,1):"-.. HIJKL       ... ..  |~ -..-..              ..  ..  }. -..-..'-..'-..'-..'-..'-..'-..'-..'-..'-"
330 GOSUB 1680
350 GOSUB 1400 :: CALL SPRITE(#10,128,7,60,40,#11,132,7,76,40)
360 CALL JOYST(1,A,B) :: CALL KEY(1,N,S) :: IF S<>0 THEN GOSUB 1000
370 IF B=4 THEN 2000 ELSE IF A THEN GOSUB 2500
380 CALL POSITION(#10,Y,X) :: IF X<40 THEN CALL LOCATE(#10,Y,40,#11,Y+16,40) ELSE IF X>235 THEN 700
390 CALL GCHAR(INT((Y+16)/8+2),INT(X/8)+1,G) :: IF G=80 OR G=81 THEN 14000
400 GOSUB 29000 :: GOTO 360
700 SCORE=SCORE+100 :: BILD=BILD-1 :: GOSUB 1400 :: IF BILD<=0 THEN 800 ELSE CALL DELSPRITE(ALL) :: GOTO 350
800 WAVE=2 :: CALL DELSPRITE(ALL) :: GOSUB 1680 :: CALL FIGUREN(2) :: DAM=10 :: SCORE=SCORE+500 :: GOSUB 1400
810 DISPLAY AT(10,1):"(*)(*)(*)()()
()*(*)***)***))EFCCCCCCCGCCCCCCBCCCCCCCCGCGCCGCGCGCGGEFEFCGCBCBCBCCCCCADADAADADADDAAAADDADDDDADADA"
820 DISPLAY AT(14,1):"PPQPQPQPQPQPQPQPQPQPQPPQPQPQPQPPPPPPPQQQQQQPQPQPPQPQPPPQPQPQPQPQPQPQPPPPPPPQPQPQP"
840 CALL SPRITE(#10,128,7,60,40,#11,132,7,76,40,#20,104,8,55,250,0,-7)
850 CALL JOYST(1,A,B) :: CALL KEY(1,N,S) :: IF S<>0 THEN GOSUB 1000
860 CALL PATTERN(#20,100) :: IF A THEN GOSUB 2500
870 IF INT(RND*5+1)=5 THEN 5000
880 CALL COINC(#10,#20,10,C) :: CALL COINC(#11,#7,20,C1) :: IF C1<>0 OR C<>0 THEN 14000
885 CALL PATTERN(#20,104)
890 GOTO 850
1000 CALL POSITION(#10,Y,X) :: IF X>230 THEN RETURN
1005 CALL PATTERN(#10,120) :: CALL SPRITE(#12,116,7,Y,X+16) :: CALL SOUND(-20,-5,3) :: CALL COINC(#12,#20,7,C)
1010 IF C THEN CALL PATTERN(#10,128) :: CALL DELSPRITE(#12) :: GOTO 1100 ELSE CALL COINC(#10,#20,10,C) :: IF C THEN 14000
1020 CALL PATTERN(#10,128) :: CALL DELSPRITE(#12) :: RETURN
1100 IF WAVE=3 THEN 10000
1105 CALL MOTION(#20,0,0) :: CALL POSITION(#20,Y1,X1) :: CALL SOUND(-250,110,30,110,30,10000,30,-8,4) :: CALL SPRITE(#19,92,9,Y1,X1)
1110 DAM=DAM-1 :: IF DAM<=0 THEN 1200 ELSE SCORE=SCORE+200 :: GOSUB 1400 :: CALL DELSPRITE(#19) :: CALL MOTION(#20,0,80)
1115 FOR I=1 TO 50 :: NEXT I
1120 CALL MOTION(#20,0,-7) :: IF WAVE=2 THEN 850
1200 CALL DELSPRITE(#19) :: CALL MOTION(#20,5,2) :: FOR I=22 TO 1 STEP -1 :: CALL PATTERN(#20,104)
1205 CALL SOUND(-250,I*110,5,I*110,5) :: CALL PATTERN(#20,100) :: NEXT I
1210 CALL MOTION(#20,0,0) :: CALL POSITION(#20,Y1,X1) :: FOR I=1 TO 4 :: CALL SOUND(-250,110,30,110,30,I*1000,30,-8,3)
1220 CALL SPRITE(#I,96,10,Y1+INT(RND*-6+3),X1+INT(RND*-6+3)) :: NEXT I :: CALL DELSPRITE(#20)
1230 FOR I=1 TO 7 :: FOR J=3 TO 1 STEP -1 :: CALL SOUND(-1500,110,30,110,30,J*500,30,-8,I*2) :: NEXT J :: NEXT I
1240 CALL MOTION(#1,-50,-50,#2,50,50,#3,50,-50,#4,-50,50) :: SCORE=SCORE+1000 :: GOSUB 1400 :: CALL DELSPRITE(#1,#2,#3,#4) :: GOSUB
1680
1250 CALL DELSPRITE(ALL) :: GOTO 3000
1400 DISPLAY AT(20,9) SIZE(7):SCORE :: DISPLAY AT(22,9) SIZE(4):DAMAGE*10 :: RETURN
1610 FOR I=1 TO 10 :: CALL SOUND(-500,247,I,494,I,123,I*3) :: NEXT I :: CALL SOUND(250,262,3) :: CALL SOUND(250,294,1,584,10)
1620 CALL SOUND(300,330,0,330,0,659,5) :: CALL SOUND(250,294,2,587,10) :: CALL SOUND(200,262,3,523,10) :: FOR I=1 TO 15
1630 CALL SOUND(-500,330,I,330,I+5,659,I*2) :: NEXT I :: CALL SOUND(200,330,4,659,4) :: CALL SOUND(200,349,4,698,4) ::
1640 CALL SOUND(200,330,4,659,4) :: CALL SOUND(200,294,3,587,3) :: CALL SOUND(200,262,3,523,3)
1650 CALL SOUND(100,294,4,587,6) :: CALL SOUND(100,262,4,523,6) :: FOR I=1 TO 20 :: CALL SOUND(-500,247,I,494,I+4,988,I+10) :: NEXT I
1660 RETURN
1680 CALL SOUND(400,262,2) :: CALL SOUND(200,277,3,554,8) :: CALL SOUND(200,311,2,622,5) :: CALL SOUND(400,415,0,415,0,831,3)
1690 CALL SOUND(200,262,2,523,5) :: CALL SOUND(200,277,2,554,5) :: FOR I=1 TO 10 :: CALL SOUND(-500,311,I,311,I*2,622,I*3) :: NEXT I
1700 RETURN
2000 CALL POSITION(#10,Y,X) :: IF X>230 THEN 360 ELSE CALL PATTERN(#10,140) :: CALL SOUND(-1000,-6,10) :: CALL
SPRITE(#12,112,7,Y-10,X+10)
2010 FOR I=0 TO 20 :: CALL SOUND(-10,-5,I) :: NEXT I :: CALL SPRITE(#13,112,7,Y-26,X+26) :: CALL PATTERN(#11,124) :: CALL
PATTERN(#13,108)
2020 CALL LOCATE(#12,Y-10,X+18) :: CALL PATTERN(#12,108) :: CALL LOCATE(#10,Y-5,X+12,#11,Y+11,X+12)
2030 GOSUB 29000 :: CALL COINC(#11,#20,10,C) :: IF C THEN 14000 ELSE IF X+24>235 THEN 700
2040 CALL PATTERN(#13,104) :: CALL LOCATE(#13,Y-26,X+34,#12,Y-10,X+42) :: CALL PATTERN(#12,104) :: CALL
LOCATE(#10,Y-5,X+34,#11,Y+11,X+34)
2050 GOSUB 29000 :: CALL COINC(#11,#20,10,C) :: IF C THEN 14000 ELSE IF X+34>235 THEN 700
2060 CALL PATTERN(#13,100,#12,100) :: CALL LOCATE(#12,Y-10,X+50,#10,Y-5,X+42,#11,Y+11,X+42)
2070 GOSUB 29000 :: CALL COINC(#11,#20,10,C) :: IF C THEN 14000 ELSE IF X+32>235 THEN 700
2080 CALL PATTERN(#11,132) :: CALL DELSPRITE(#12,#13) :: CALL LOCATE(#10,Y,X+55,#11,Y+16,X+55) :: CALL PATTERN(#10,128) ::
SCORE=SCORE+20
2090 GOSUB 1400 :: GOTO 380
2500 CALL MOTION(#10,0,A*2,#11,0,A*2) :: CALL SOUND(-10,-5,5) :: CALL PATTERN(#11,136) :: FOR I=1 TO 20 :: NEXT I :: CALL SOUND(-10,-
5,5)
2505 CALL PATTERN(#11,132)
2510 FOR I=1 TO 20 :: NEXT I :: CALL MOTION(#10,0,0,#11,0,0) :: CALL POSITION(#10,Y,X) :: CALL LOCATE(#11,Y+16,X) :: RETURN
3000 CALL DELSPRITE(ALL) :: DISPLAY AT(14,1):RPT$("EF",56)
3001 CALL CHAR(77,"FFFF0602030203060D0A0D0B346BD7FFFFFF60C040C040E0F0F0F0D01CFEFFFF")
3010 CALL COLOR(6,15,1,7,15,1) :: CALL HCHAR(9,18,77) :: CALL HCHAR(10,18,78) :: CALL HCHAR(9,19,79) :: CALL HCHAR(10,19,80) :: CALL
FIGUREN(3)
3020 WAVE=3 :: SCORE=SCORE+1000 :: DAM=10 :: IF DAMAGE>=8 THEN DAMAGE=DAMAGE-1
3030 GOSUB 1400
3040 CALL SPRITE(#10,128,7,50,30,#11,132,7,66,30,#20,96,10,50,230,#21,108,7,66,230)
3050 CALL JOYST(1,A,B) :: CALL KEY(1,N,S) :: IF S<>0 THEN GOSUB 1000 ELSE IF A THEN GOSUB 2500
3060 IF INT(RND*2+1)=2 THEN 11000
3070 IF INT(RND*4+1)=3 THEN 12000
3080 GOTO 3050
5000 FOR I=10000 TO 1000 STEP -2000 :: CALL SOUND(-500,I,I/1000*2,I+5000,I/1000*2) :: CALL MOTION(#20,0,I/1000-1) :: NEXT I
5010 CALL SOUND(-500,111,30,111,30,1000,30,-8,4) :: CALL POSITION(#20,Y1,X1) :: CALL SPRITE(#21,108,9,Y1,X1-8,7,-9)
5020 FOR I=1 TO 15 :: CALL COINC(#10,#21,10,C) :: IF C THEN 14000
5025 NEXT I :: CALL POSITION(#21,Y1,X1) :: CALL DELSPRITE(#21)
5030 CALL MOTION(#20,0,-10) :: CALL SOUND(-1000,111,30,111,30,500,30,-8,0) :: CALL SPRITE(#7,96,11,Y1,X1)
5040 GOTO 880
10000 CALL POSITION(#20,Y1,X1) :: CALL SPRITE(#18,92,11,Y1+7,X1) :: CALL SOUND(-10,300,30,-5,5) :: DAM=DAM-1 :: IF DAM<=0 THEN 10100
10010 SCORE=SCORE+200 :: GOSUB 1400 :: CALL DELSPRITE(#18) :: :: CALL MOTION(#20,0,20,#21,0,20) :: FOR I=1 TO 20 :: NEXT I
10020 CALL MOTION(#20,0,0,#21,0,0) :: CALL POSITION(#20,Y1,X1) :: CALL LOCATE(#21,Y1+16,X1) :: GOTO 3050
10100 CALL DELSPRITE(#18)
10105 CALL CHAR(112,"0002070F1F1F1F0B0B0B07050301010000080C0C0808080800000000000000")
10120 FOR I=1 TO 10 :: CALL SCREEN(16) :: CALL SCREEN(2) :: NEXT I :: CALL SPRITE(#15,112,16,48,140)
10130 GOSUB 1680 :: SCORE=SCORE+2000 :: GOSUB 1400
10140 CALL JOYST(1,A,B) :: IF A THEN GOSUB 2500
10150 CALL COINC(#10,#15,6,C) :: IF C THEN 10160 ELSE 10140
10160 CALL POSITION(#10,Y,X) :: CALL PATTERN(#10,120) :: CALL SPRITE(#12,116,7,Y,X+16) :: CALL LOCATE(#15,Y-4,X+16) :: GOSUB 1610
10180 CALL COLOR(#15,14)
10230 CALL CHAR(96,"0303020103000103010700070003014 3C0C0C04080C0C0A0D0F0D0F0D0F0D0F0F0D090")
10240 CALL CHAR(104,"3F7E21030306060C0C1C18180C0C0C3C2078F87060606060E060603030181878")
10260 CALL COLOR(#20,16,#21,16) :: FOR I=1 TO 100 :: NEXT I :: FOR I=1 TO 4 :: CALL SPRITE(#I,92,16,Y1+8+INT(RND*-6+3),X1)
```

```
10265 NEXT I :: CALL SOUND(-2000,110,30,110,30,300,30,-8,0)
10270 CALL MOTION(#20,-30,0,#21,50,0,#1,-40,-40,#2,40,-40,#3,-40,40,#4,40,40)
10280 FOR I=1 TO 100 :: NEXT I :: CALL DELSPRITE(#1,#2,#3,#4,#20,#21) :: FOR I=DAMAGE TO 0 STEP -0..5 :: DAMAGE=I :: GOSUB 1400 ::
NEXT I
10290 CALL DELSPRITE(ALL) :: CALL CLEAR :: CALL CHARSET :: FOR I=1 TO 8 :: CALL COLOR(I,7,1) :: NEXT I
10295 SC$=STR$(SCORE) :: L=LEN(SC$) :: CALL CHAR(63,RPT$("0",16-L)&SC$)
10300 DISPLAY AT(1,1):"INDIANA JONES": :"(C) BY THOMAS DREIBRODT": : :"   NOCHEINMAL SPIELEN": :"   SPIEL 1 LADEN" :: M=6
10305 CALL VCHAR(5,3,32,8) :: CALL HCHAR(M,3,42) :: CALL JOYST(1,A,B) :: IF B=4 THEN M=6 ELSE IF B=-4 THEN M=8
10310 CALL KEY(1,N,S) :: IF S=0 THEN 10305 ELSE IF M=6 THEN CALL CLEAR :: RUN ELSE IF M=8 THEN PRINT "* NACHLADEN,BITTE WARTEN....."
:: RUN "CS1"
11000 CALL MOTION(#20,0,-5,#21,0,-5) :: CALL PATTERN(#21,108) :: CALL SOUND(10,220,20,-5,5) :: FOR I=1 TO 20 :: NEXT I :: CALL
PATTERN(#21,104)
11010 CALL SOUND(10,110,20,-5,5) :: FOR I=1 TO 20 :: NEXT I :: CALL MOTION(#20,0,0,#21,0,0) :: CALL POSITION(#20,Y1,X1) :: CALL
LOCATE(#21,Y1+16,X1)
11020 GOTO 3070
12000 CALL POSITION(#20,Y1,X1) :: IF X<30 THEN 3050 ELSE CALL PATTERN(#20,100) :: CALL SPRITE(#19,112,10,Y1,X1-16) :: CALL
SOUND(-20,-5,3)
12010 CALL COINC(#19,#10,8,C) :: IF C THEN CALL PATTERN(#20,96) :: CALL DELSPRITE(#19) :: GOTO 14000
12020 FOR I=1 TO 20 :: NEXT I :: CALL DELSPRITE(#19) :: CALL PATTERN(#20,96) :: GOTO 3050
14000 CALL POSITION(#10,Y,X)
14010 CALL PATTERN(#11,124) :: CALL LOCATE(#10,Y+5,X) :: CALL SPRITE(#9,92,9,Y+10,X) :: CALL SOUND(50,-5,3) :: FOR I=1 TO 100 :: NEXT
I
14011 CALL DELSPRITE(#12,#13,#20,#21,#7)
14015 DAMAGE=DAMAGE+..5 :: GOSUB 1400 :: IF DAMAGE>=10 THEN 14100
14020 CALL DELSPRITE(#9) :: CALL LOCATE(#10,Y,X) :: CALL PATTERN(#11,128) :: IF WAVE=1 THEN 350 ELSE IF WAVE=2 THEN 840 ELSE IF
WAVE=3 THEN 3040
14100 CALL LOCATE(#10,Y,X) :: FOR I=9 TO 1 STEP -1 :: CALL SOUND(-1500,111,30,111,30,I*500,30,-8,0) :: CALL COLOR(#10,7,#11,7)
14110 CALL PATTERN(#10,128,#11,132) :: CALL SPRITE(#I,92,9,Y+5,X,INT(RND*-8)+4,INT(RND*-6)+3) :: CALL PATTERN(#10,84,#11,88)
14120 CALL COLOR(#10,16,#11,16) :: NEXT I :: FOR I=1 TO 9 :: CALL DELSPRITE(#I) :: NEXT I :: FOR I=1 TO 100 :: NEXT I :: GOSUB 1610
14130 CALL KEY(1,N,S) :: IF S THEN CALL DELSPRITE(ALL) :: RUN ELSE 14130
29000 IF INT(RND*3+1)=3 THEN CALL SOUND(-500,-6,5) :: CALL SPRITE(#20,96,12,70,INT(RND*2+1)*80+INT(RND*-3+6))
29010 IF INT(RND*4+1)=1 THEN CALL DELSPRITE(#20) :: RETURN ELSE RETURN
30000 SUB FIGUREN(E)
30001 ON E GOTO 30002,30008,30012
30002 CALL CHAR(112,"0000000000000000010204081020408001020408102040800000000000000000")
30003 CALL CHAR(108,"0000000000000000000000000000000000010102020404080810102020404080 80")
30004 CALL CHAR(104,"8080404020201010080804040202010100000000000000000000000000000000")
30005 CALL CHAR(100,"8040201008040201000000000000000000000000000000000008040201008040201")
30006 CALL CHAR(96,"00242071722236362E7F7F7F7F3F3F1FA12026247070747CFCFCFCFCF8F8F0")
30007 SUBEXIT
30008 CALL CHAR(100,"78D878B931367D7BC5C6C566606E3F3140BFD7AD5DEA94A850A0C0000C3EF3C0")
30009 CALL CHAR(104,"78D878B831677FFDC2C1C0CCDF7360000000607FAF75EA5CB0C2010102CEFC30")
30010 CALL CHAR(108,"0000000021020 4401C327D7D7F3E1C000F37572E54B458A85050204008000040")
30011 SUBEXIT
30012 CALL CHAR(96,"0303060303091133E3E7F7DF1E0E0E06C0E0E0E0C0C0F0B0B0B07878F8ECEC6C")
30013 CALL CHAR(100,"1E1F371F1E07070FFFFDFD1F0F0D0D0D0000000000C0E0F0B0D8D8B8BCBCBCDC")
30014 CALL CHAR(104,"0307040707070F0F0F1F1F0C0C0C3C7CECF436F6FFBEBC7C7EFF3F0606060E3E")
30015 CALL CHAR(108,"0E0F081F3F767E6EF7FFFC7E3E3E270F18ECF434F8F8F8F8FC7C7E3E3C0C3C7C")
30016 CALL CHAR(112,"000000000000000000030F000000000000000000000000007FFFF3800000000")
30017 SUBEND
```

jager des verlorenen schatz

Basic-Kurs Teil 1

BASIC für den Anfänger!

Einleitung

Dieser Basic-Kurs ist für Nicht-Mathematiker, Nicht-Techniker, Nicht-Computerprofis geschrieben. Er soll ein Leitfaden für Anfänger sein, die sich der Programmierung ihres eventuell auch zukünftigen Heimcomputers widmen wollen und dazu keine besonderen Voraussetzungen mitbringen als den Wunsch, eigene Programme (z. B. Spiele) auszutüfteln. Aus diesem Grund wurde besonders auf die Verständlichkeit und Vollständigkeit dieses Kurses Wert gelegt.
Der Kurs ist so aufgebaut, das der Leser so rasch wie möglich befähigt wird, eigene kleine Programme zu schreiben. Der beschriebene Basic-Umfang orientiert sich an den zur Zeit verbreiteten Heimcomputern wie Commodore C-64, VC-20, C-16, Apple, Spectrum, MSX und CPC 464, soweit es sich nicht um Details handelt, die nur von den technischen Gegebenheiten einzelner Typen abhängen.

Was bedeutet eigentlich das Wort „Programmieren"?

Der Computer ist eine Maschine, die dazu konstruiert wurde, Informationen zu verarbeiten – sowohl Text- als auch Zahleninformationen. Benötigt man den Computer im Umgang mit Texten (z. B. Brief schreiben), so spricht man von der „Textverarbeitung". Geht man vorwiegend mit Zahlen um, redet man von „Zahlenverarbeitung".
Um dem Computer seine Aufgabe mitzuteilen, ist es nötig, Befehle in einem Format oder einer „Sprache" zu geben, die der Computer versteht. Jeder Computer kann nur eine ganze Anzahl von verschiedenen Befehlen (ca. 100 – 500) verstehen. Befehle, die der Computer direkt verstehen kann, werden Maschinensprache-Befehle genannt. Diese bestehen nur aus einer oder mehreren Zahlen, welche im Speicher des Computers abgelegt werden.
Eine Folge von Befehlen, die etwas Nützliches bewirken, ist ein „Programm". Ein Computer befolgt ein Programm, indem er Befehl für Befehl ausführt. Ideal wäre es, wenn man die Befehle in menschlicher Umgangssprache an den Computer weitergeben könnte. Das ist aber nicht möglich, da der Computer keine gewöhnlichen Sprachen versteht – weder gesprochen noch geschrieben. Der Grund: Ein Computer führt Anweisungen genau und exakt aus. Er arbeitet logisch und präzise, und er verlangt klare und eindeutige Befehle in der richtigen Reihenfolge und Form.
Aus diesem Grund sind vereinfachte „Sprachen" für den Umgang mit Computern erfunden wurden. Man spricht von „höheren Sprachen", welche als eine Art Übersetzer (Interpreter) zwischen Benutzer und der eigentlichen Maschinensprache des Computers arbeiten.
Für eine klare Verständigung mit einem Computer kann nur eine begrenzte Anzahl von englischen Wörtern benutzt werden, die als genau festgelegte Kommandos verwendet werden. Außerdem müssen sich Anweisungen und Befehls-Folgen, die den Computer steuern sollen, ganz genau an bestimmte Grammatikregeln (genannt Syntax-Regeln) halten. Die Kombination aus einem begrenzten Wortschatz und einer Syntax wird Programmiersprache genannt. *„BASIC"* ist eine solche höhere Programmiersprache.

Algorithmus – Programmablaufplan

Programmieren ist eine interessante und spannende Tätigkeit geworden, eine Tätigkeit, die den weiten Bereich zwischen Handwerk und Kunst einnimmt. Wir wollen nun mit einigen wichtigen Dingen beginnen, die mit der handwerklichen Seite des Programmierens zu tun haben.
Eines der wichtigsten Hilfsmittel von Programmierern ist der Programmablaufplan (Flußdiagramm).
Gleichgültig, in welcher Programmiersprache wir programmieren, die Technik des Aufstellens eines solchen Flußdiagramms ist immer die gleiche:
1. Suche nach einem Lösungsweg (Algorithmus)
2. Aufbau des Flußdiagramms nach Norm (DIN 44300)
Jedes Programm kann zeichnerisch als Programmablaufplan (PAP) dargestellt werden. Der Programmablaufplan, auch Flußdiagramm genannt, besteht aus folgenden nach DIN 44300 normierten Sinnbildern, mit denen alle

weiter Seite 16

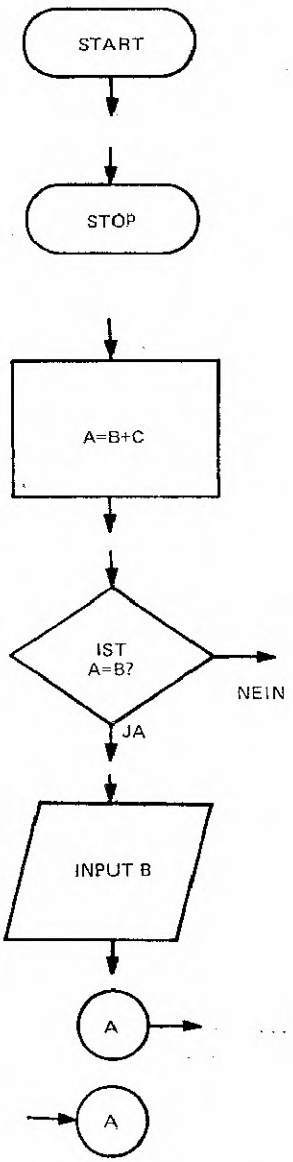
Kurs

Fortsetzung von Seite 7

in einem Programmablauf möglichen Wege dargestellt werden.

Die wichtigsten Symbole sind:

— Bild 1 —



Grenzstelle: Anfang des Programmablaufplans.

Grenzstelle: Ende des Programmablaufplans.

Allgemeine Operation. Dieses Kästchen kennzeichnet irgendwelche Tätigkeiten, hier z. B. 'Addition von B und C. Zuweisung des Resultats zu A'.

Verzweigung

Eingabe, z. B. über die Tastatur. Auch Ausgabe.

Übergangsstelle. Wenn der Ablaufplan z. B. auf der nächsten Seite weitergeht, dann verwendet man zwei solcher Symbole.

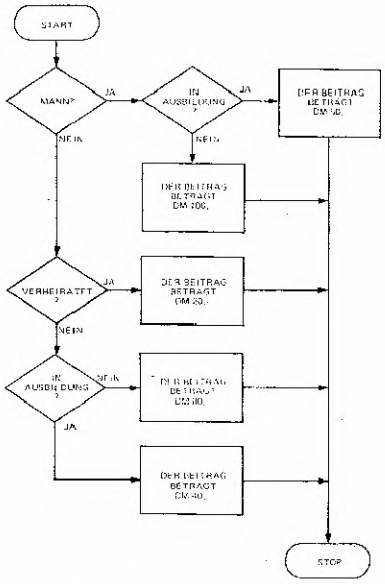
Das Flußdiagramm ist für umfangreiche Programme unerläßlich. Dem Profi-Programmierer hilft es rechtzeitig vor dem Schreiben des eigentlichen Programms, Fehler und Ungereimtheiten aufzuspüren. Außerdem erspart ein Flußdiagramm eine umständliche und langwierige Fehlersuche im fertigen Programm. Wir wollen nun an Hand eines kleinen Beispieles ein Flußdiagramm erstellen.
Die Aufgabe des Programmes besteht darin, den Beitrag für ein neues Clubmitglied festzulegen.

Als Ausgangsdaten sind vorhanden:
Geschlecht: männlich
Ausbildung: ja

Unser Programm arbeitet nach folgenden Algorithmus:
Der Jahresbeitrag für einen Mann beträgt 100,- DM. Ist er in der Ausbildung, dann beträgt der Jahresbeitrag nur die Hälfte, also 50,- DM. Der Jahresbetrag für eine Frau beträgt nur 80,- DM. Ist diese noch in der Ausbildung, so beträgt der Jahresbeitrag nur 40,- DM.

Dieses Beispiel ist relativ einfach und bedarf in der Regel kein Flußdiagramm. Wir haben dieses nur zur Übung aufgezeichnet:

— Bild 2 —



Sind Sie eine verheiratete Frau, so genügt ein kurzer Blick auf Bild 2, um die Fragen „Sind Sie ein Mann" und „Sind Sie verheiratet" zu beantworten.
Aus diesem Beispiel kann man sehen, wie ein Ablaufplan dabei hilft, einen auf den ersten Blick komplizierten Sachverhalt zu vereinfachen.

Funktion und Vokabular von Basic

Eigenschaften:
Die höhere Programmiersprache Basic ist die am weitesten verbreitete Sprache für den Computer. Sie wurde 1964 am College of Dartmouth in New Hampshire (USA) entwickelt. Das Wort „Basic" steht für die Abkürzung „The Beginners All Purpose Symbolic Instruction Code". Das bedeutet etwa: „allgemein einsetzbarer, symbolischer Anweisungscode für Anfänger". Basic ist jedoch weder leichter noch schwieriger zu programmieren. Jedoch werden viele Befehle aus der englischen Umgangssprache verwendet, wie z. B. GOTO, READ, RUN usw.

Basic-Interpreter:
Basic ist eine interpretative Sprache, was bedeutet, daß in Basic geschriebene Programme beim Ablauf in Maschinensprache umgesetzt werden. Diese Methode erspart einen zusätzlichen Übersetzer. Neben dem Vorteil, daß

Basic auch in kleinen und billigen Rechnern eingesetzt werden kann, ermöglicht dies auch eine schnelle Korrektur von Programmfehlern.
Nachteilig ist die durch die Übersetzung verursachte lange Ausführungszeit.

Basic-Compiler:
Obwohl Basic eigentlich eine Interpreter-Sprache ist, gibt es mittlerweile schon mehrere Basic-Compiler auf dem Markt. Dieser übersetzt das eigentliche Basic-Programm in Maschinensprache oder in einen speziellen Zwischencode. Durch eine Compilierung erreicht man je nach Compiler eine 2- bis 10fache Geschwindigkeitserhöhung.

Die ersten Schritte
Wie bei den natürlichen Sprachen gibt es auch bei den Programmiersprachen viele Abarten. Auch Basic hat viele verschiedene „Dialekte".
Fast jeder Hersteller von Computern hat eine eigene Version von Basic entwickelt. Es gibt jedoch einen bestimmten Grundwortschatz, welcher in jeder Version von Basic zu finden ist.

Grundwortschatz in Basic:

— Bild 3 —

BASIC-Schlüsselwörter:

ABS	IF	RETURN
ATN	INPUT	RND
COS	INT	SGN
DATA	LET	SIN
DEF	LOG	SQR
DIM	NEXT	STEP
END	ON	STOP
EXP	PRINT	TAB
FOR	READ	TAN
GOSUB	REM	THEN
GOTO	RESTORE	

Ehe wir jedoch ans Programmieren gehen, müssen wir uns erst mit einigen Begriffen vertraut machen, um diese Einführung in die Programmiersprache Basic richtig verstehen zu können.

Zeichensatz (ASCII-Norm)
Neben Zahlen besitzt ein Computer noch zahlreiche weitere Zeichen. Der Zeichenvorrat von Basic ist genormt und besitzt folgende Zeichen:
26 Großbuchstaben (A–Z)
10 Ziffern (0–9)
Sonderzeichen
Steuercodes

Unterschiedliche Basic-Versionen können einen unterschiedlichen Vorrat an Sonderzeichen besitzen.
Als praktische Hilfe hier eine Zusammenstellung des Standard-Ascii-Zeichensatzes mit dezimaler, oktaler und hexadezimaler Schreibweise.

— Bild 4 —

DEC	OCTAL	HEX	ASCII characters	DEC	OCTAL	HEX	ASCII	DEC	OCTAL	HEX	ASCII
0	000	00	NUL ([CTRL]A)	50	062	32	2	100	144	64	d
1	001	01	SOH ([CTRL]B)	51	063	33	3	101	145	65	e
2	002	02	STX ([CTRL]C)	52	064	34	4	102	146	66	f
3	003	03	ETX ([CTRL]C)	53	065	35	5	103	147	67	g
4	004	04	EOT ([CTRL]D)	54	066	36	6	104	150	68	h
5	005	05	ENQ ([CTRL]E)	55	067	37	7	105	151	69	i
6	006	06	ACK ([CTRL]F)	56	070	38	8	106	152	6A	j
7	007	07	BEL ([CTRL]G)	57	071	39	9	107	153	6B	k
8	010	08	BS ([CTRL]H)	58	072	3A	:	108	154	6C	l
9	011	09	HT ([CTRL]I)	59	073	3B	;	109	155	6D	m
10	012	OA	LF ([CTRL]J)	60	074	3C	<	110	156	6E	n
11	013	OB	VT ([CTRL]K)	61	075	3D	=	111	157	6F	o
12	014	OC	FF ([CTRL]L)	62	076	3E	>	112	160	70	p
13	015	OD	CR ([CTRL]M)	63	077	3F	?	113	161	71	q
14	016	OE	SO ([CTRL]N)	64	100	40	@	114	162	72	r
15	017	OF	SI ([CTRL]O)	65	101	41	A	115	163	73	s
16	020	10	DLE ([CTRL]P)	66	102	42	B	116	164	74	t
17	021	11	DC1 ([CTRL]Q)	67	103	43	C	117	165	75	u
18	022	12	DC2 ([CTRL]R)	68	104	44	D	118	166	76	v
19	023	13	DC3 ([CTRL]S)	69	105	45	E	119	167	77	w
20	024	14	DC4 ([CTRL]T)	70	106	46	F	120	170	78	x
21	025	15	NAK ([CTRL]U)	71	107	47	G	121	171	79	y
22	026	16	SYN ([CTRL]V)	72	110	48	H	122	172	7A	z
23	027	17	ETB ([CTRL]W)	73	111	49	I	123	173	7B	{
24	030	18	CAN ([CTRL]X)	74	112	4A	J	124	174	7C	|
25	031	19	EM ([CTRL]Y)	75	113	4B	K	125	175	7D	}
26	032	1A	SUB ([CTRL]Z)	76	114	4C	L	126	176	7E	~
27	033	1B	ESC	77	115	4D	M				
28	034	1C	FS	78	116	4E	N				
29	035	1D	GS	79	117	4F	O				
30	036	1E	RS	80	120	50	P				
31	037	1F	US	81	121	51	Q				
32	040	20	SP	82	122	52	R				
33	041	21	!	83	123	53	S				
34	042	22	"	84	124	54	T				
35	043	23	#	85	125	55	U				
36	044	24	$	86	126	56	V				
37	045	25	%	87	127	57	W				
38	046	26	&	88	130	58	X				
39	047	27	'	89	131	59	Y				
40	050	28	(	90	132	5A	Z				
41	051	29	)	91	133	5B	[				
42	052	2A	*	92	134	5C	\				
43	053	2B	+	93	135	5D	]				
44	054	2C	,	94	136	5E	↑				
45	055	2D	-	95	137	5F	_				
46	056	2E	.	96	140	60					
47	057	2F	/	97	141	61	a				
48	060	30	Ø	98	142	62	b				
49	061	31	1	99	143	63	c				

Was sind Konstanten?

In Basic unterscheidet man zwei Arten von Konstanten, ganzzahlige oder reelle.
Ganzzahlige Konstanten (Integer) können nur einen Wert ohne Nachkommastelle annehmen. Reele Konstanten dagegen können auch Variablen mit Nachkommastellen verarbeiten.
Hierbei ist eine Eigentümlichkeit zu beachten. Reelle Konstanten schreibt man in Basic nicht mit einem Dezimalkomma, sondern mit einem Dezimalpunkt:
falsch: 3,14 – richtig: 3.14
falsch: –6,0 – richtig: –6.0
Für Dezimalzahlen gibt es in Basic noch eine andere Schreibweise. Wir wissen vielleicht noch aus dem Mathematikunterricht, daß man z. B. statt 6543,2 auch 6,5432 x 10 ^{3} schreiben kann.
Man spricht dann von der technisch wissenschaftlichen Notation. Da Basic keine hochgestellten Zeichen kennt, wurde hier das Zeichen „E" für Exponent gewählt.
Das bedeutet, daß die Zahl 6543,2 auf zwei verschiedene Arten geschrieben werden kann:
entweder 6543,2 oder 6.5432 E3
So, für dieses mal soll es genug sein. Im nächsten Teil beginnen wir, erste kleine Programme zu entwickeln.

Frank Brall
Elke Reif

Verwendete Literatur:
Grundkurs in Basic, Ulrich Stroebel, Sybex-Verlag. Basic Brevier, Siegmar Wittig, Heinz Heise GmbH. ZX-Spectrum Handbuch, Steven Vickers, Sinclair. Commodore 116/16/plus 4, Ekkehard Kaier, Vieweg-Verlag. Schneider CPC 464, Rodnay Zaks, Sybex-Verlag

Basic-Kurs – Teil 2

BASIC für Anfänger!

Variablen!

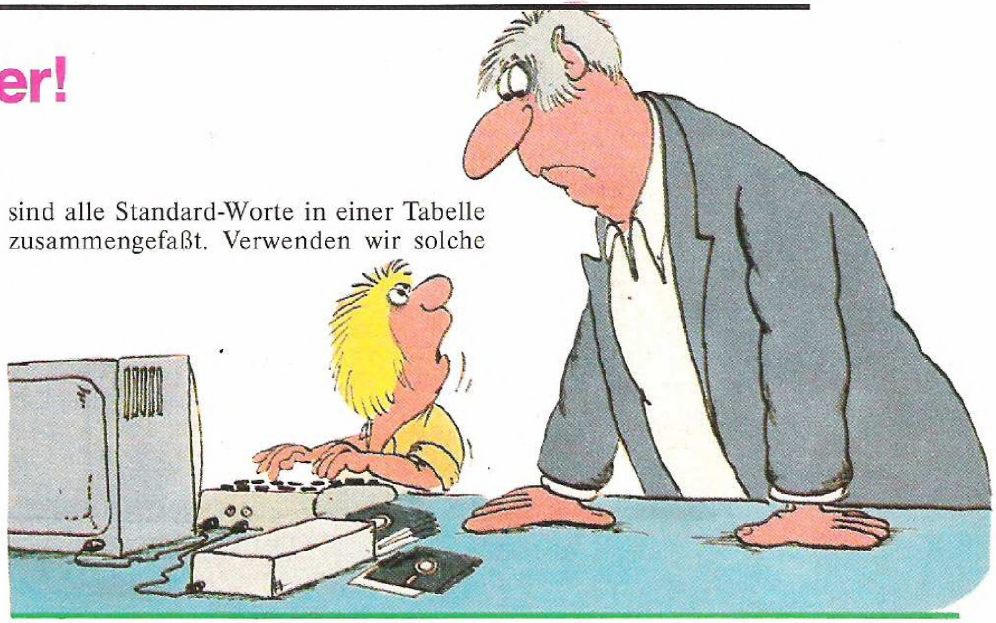
Eine Variable hat einen Namen, einen Datentyp und einen Wert, der sich ändern kann und somit variabel ist.
Basic unterscheidet die drei Datentypen: Integer, Real und String
Der jeweilige Datentyp wird durch ein Typzeichen festgelegt. Geben wir keines der beiden Datentypzeichen % und $ am Ende eines Variablennamens an, dann nimmt Basic automatisch Real als Datentyp an. Mit Ausnahme des Typzeichens können wir unsere Variablennamen beliebig wählen. Dabei müssen sie sich in den ersten zwei Zeichen unterscheiden, da diese nur vom Computer erkannt werden. Einige Computer können auch noch längere Variablennamen verarbeiten.

Zulässige Variablen-Länge bei unterschiedlichen Variablen:

Commodore	2 Zeichen
Schneider CPC	beliebige Anzahl
Spectrum	beliebige Anzahl
ZX 81	beliebige Anzahl
MSX	2 Zeichen
Apple	2 Zeichen
Atari	beliebige Anzahl
TI 99	beliebige Anzahl

Computer, welche nur 2 Zeichen von den Variablen untersuchen, arbeiten gewöhnlich etwas schneller, da die Untersuchung der einzelnen Variablen einfacher ist. Der Vorteil von längeren Variablennahmen ist, daß man die Bezeichnung beliebig wählen und so übersichtlichere Programme entwickeln kann.
Die Namen MEHRWERTSTEUER, ME7715$, MELDUNG% und MEERESTIEFE werden somit in Commodore-Basic als ME, ME, ME$, ME, ME% und ME unterschieden.
Viermal taucht der Variablenname ME auf; solche Überschneidungen führen in Basic häufig zu Fehlern. Man sollte solche Bezeichnungen möglichst vermeiden.

Eine weitere Fehlerquelle sind die in Basic reservierten Worte wie LIST, FN, GOSUB, TO, NEXT oder PRINT. Im letzten Teil sind alle Standard-Worte in einer Tabelle zusammengefaßt. Verwenden wir solche



Worte als Variablennamen, so führt dieses zwangsläufig zu Fehlern. Der Computer kann in einem solchen Fall nicht feststellen, ob eine Variable oder ein Befehl gemeint ist. Er gibt eine Fehlermeldung aus.

Das Einrichten bzw. Definieren von Variablen heißt VEREINBARUNG. Dabei teilt man durch Angabe des Typzeichens den Datentyp mit.

Datentype	Typezeichen	Beispielname
Integer	%	A%
REAL	kein Zeich.	A
String	$	A$

Wir lernen die ersten Befehle kennen !

LET

Dies ist wohl der einfachste BASIC-Befehl, welcher in fast allen Computern verwendbar ist. Er gestattet auf einfachste Weise die Definition von Variablen.

Anwendungsbeispiele von LET:

Beispiel 1	LET A = 10
Beispiel 2	LET A = B – 2
Beispiel 3	LET A = A – 2

Die LET-Anweisung wird meist auch als „Zuweisung" bezeichnet. Die meisten Computer besitzen zwar diese Anweisung, können jedoch auch Zuweisungen ausführen ohne den LET-Befehl.

Zu bemerken ist, daß das Gleichheitszeiche in der Zuweisung nicht „GLEICH" im üblichen Sinne des Wortes heißt, sondern soviel bedeutet wie: „Wird ersetzt durch den Wert von".
Im Beispiel 1 würde der Variablen A ein Wert von 10 zugewiesen.
Im Beispiel 2 erhält die Variable A den errechneten Wert von B minus 2.
Wir würden diese Aufgabe so schreiben:
B - 2 = A
Die Anweisung lautet in Basic jedoch
A = B - 2

Einem mathematisch geschulten Leser stehen hier sicherlich die Haare zu Berge! Wenn die Speicherstelle für die Variable A vor der Ausführung in Beispiel 3 z.B. den Wert 20 hat, dann hätte die gleiche Speicherstelle nach Ausführung den Wert 18.

An dieser Stelle sei darauf hingewiesen, daß der Computer die Anweisung sozusagen von rechts nach links im Computer verarbeitet. Der rechts vom Gleichheitszeichen stehende Teil der Zuweisung wird zuerst berechnet, und danach wird das Ergebnis zur linken Seite weitergeleitet, also der Variablen zugeordnet.
Kommt in der Anweisung A = A + 1 die Variable A das erste Mal vor, so verhalten sich die einzelnen Computer unterschiedlich. Einige Computer nehmen in diesem Fall an, daß die Variable A den Wert 0 besitzt, andere geben eine Fehlermeldung.

PRINT

Dies ist wohl der wichtigste Basic-Befehl. Bis jetzt haben wir Basic-Variablen Werte zugewiesen oder diese verändert. Aber noch kein einziger Wert wurde uns vom Computer ausgegeben. Das wollen wir hier nachholen und uns jetzt mit der wichtigsten Ausgabemöglichkeit von Basic beschäftigen.

Die auf einen Basic-Befehl folgenden Angaben werden auch als Parameter bezeichnet. Je nach Befehl benötigen wir unterschiedlich viele Parameter. Einige der Befehle kommen auch ohne Parameter aus. Das werden wir aber noch im einzelnen kennenlernen.

Beispiel für PRINT:

```
PRINT 10 + 5      <RETURN TASTE>
15                <Ausgabe vom Computer>
```

Ein Befehlswort zusammen mit einigen Parametern bezeichnet man als „Anweisung". In unserem Beispiel haben wir also die Print-Anweisung PRINT 10 + 5 eingegeben. Als Abschluß der Anweisung haben wir die Eingabetaste (RETURN oder ENTER) betätigt, um dem Interpreter zu sagen, daß er das, was wir bisher getippt haben, übernehmen und verarbeiten soll. Der Interpreter nimmt unsere Anweisung und versucht, diese in Maschinenbefehle zu setzen, die der Microprozessor dann ausführen soll.

Wie auch schon vom LET-Befehl her bekannt, verarbeitet der Computer die Anweisung von rechts. Das bedeutet: Zuerst errechnet er den Ausdruck und gibt ihn danach auf dem Bildschirm aus.

Das Befehlswort PRINT teilt dem Rechner mit, daß er etwas an die Daten-Ausgabeeinheit zu übertragen hat. Meistens ist es der Bildschirm, der mit Print adressiert und angesprochen wird. Bei einem Personal-Taschen-Computer ist dies die eingebaute LCD-Anzeige.

Zahlen und arithmetische Ausdrücke

Beinhaltet die Print-Anweisung z.B. nur einfache Zahlen, dann bleiben diese nach Ausführung der Anweisung natürlich erhalten.
Wenn wir eine Basic-Anweisung in den Computer eingeben, wird diese Anweisung direkt ausgeführt. Das heißt, daß der Computer im Direkt-Modus arbeitet.
Die Print-Anweisung erfüllt im Direkt-Modus die vollwertige Aufgabe eines Taschenrechners nach dem „EVA-Prinzip":

1. Eingabe eines arithmetischen Ausdrucks über die PRINT - Anweisung
2. Verarbeiten des Ausdrucks
3. Ausgabe des Ergebnisses auf dem Bildschirm

Hier noch einige Beispiele, welche Sie mit Ihrem Computer ausprobieren können:

```
PRINT 100 * 5          <RETURN>
500
PRINT 12 + 123         <RETURN>
145
PRINT 1234             <RETURN>
1234
PRINT 10 + 20 - 1      <RETURN>
29
PRINT 10 * 18/2        <RETURN>
90
```

Bisher haben wir die Befehle nur im Direkt-Modus ausführen lassen. Nun wollen wir richtige Programme schreiben.

Was ist eigentlich ein Programm?

Unter einem Programm versteht man eine Folge von bestimmten Befehlen, die den Computer in die Lager versetzen, ein gestecktes Ziel durch Abarbeiten der einzelnen Programmschritte zu erreichen.
Für einen Anfänger wird diese Erklärung sicher sehr kompliziert klingen. Dabei stellen die nachfolgenden Zeilen bereits ein vollständiges Programm dar!
Tippen Sie die Zeilen exakt ein. Die Zahlen am Anfang der Zeile müssen unbedingt miteingegeben werden, da es sich um die sogenannten „Zeilennummern" handelt. Anhand der Zeilennummer wird dem Computer die Reihenfolge des Programmablaufs vorgeschrieben. Wird der Zeile keine Nummer vorangestellt, so wird diese sofort interpretiert und ausgeführt: man spricht vom Direkt-Modus.
Zeilennummern dürfen je nach System nur in bestimmten Bereichen liegen:

Atari	0 bis maximal 32767
Spectrum	1 bis maximal 9999
ZX 81	1 bis maximal 9999
C 16	0 bis maximal 63999
VC 20	0 bis maximal 63999
C 64	0 bis maximal 63999
CPC 464	1 bis maximal 65535
TI 99	1 bis maximal 32767

Jede eingegebene Zeile muß durch die Return-Taste (bei einigen Systemen 'Enter') abgeschlossen werden! In unserem Kurs verwenden wir für diese Taste das Symbol „<RETURN>".

Unser erstes Programm:

```
10 LET I = 6
20 LET R = 10
30 LET U = I* R
40 PRINT U
```

Aufgabe des Programmes:

In der ersten Zeile wird der Variablen I ein Wert von 6 zugewiesen. Die Variable I steht als Formelzeichen für den Strom.
In der zweiten Zeile wird der Variablen R ein Wert von 10 zugewiesen. Diese Variable steht als Formelzeichen für den Widerstand.

In der dritten Zeile steht die Formel zur Berechnung der Spannung. Hier wird der Inhalt der Variablen I mit dem Inhalt der Variablen R multipliziert. Das Ergebnis wird der Variablen U zugewiesen.
In der letzten Zeile wird der Inhalt der Variablen U auf dem Bildschirm ausgegeben.

Haben Sie das Beispiel fertig eingegeben, so kann es durch den Befehl „RUN" gestartet werden.

```
RUN        <RETURN>
60
```

Das Ergebnis beträgt 60 Volt !

Wir wollen nun unser Programm etwas übersichtlicher gestalten:

```
10 PRINT
„BERECHNUNG DER SPANNUNG!"
20 LET I = 6
30 LET R = 10
40 LET U = I* R
50 PRINT
„DER STROM BETRÄGT:" ;I; „AMPERE"
60 PRINT
„DER WIDERSTAND BETRÄGT:" ;R; „OHM"
70 PRINT
„DIE SPANNUNG BETRÄGT:" ;U; „VOLT"
```

Dieses Programm arbeitet auf die gleiche Weise, jedoch werden hier die Werte durch Kommentare ergänzt.

In Zeile 10 wird eine neue Art des PRINT-Befehles vorgestellt. Folgt einem PRINT-Befehl ein Anführungszeichen, so wird der darauf folgende Text, bis zu einem weiterem Anführungszeichen, auf dem Bildschirm ausgegeben. In diesem Fall ist dies der Text: BERECHNUNG DER SPANNUNG! Fortsetzung auf Seite 35

BASIC für Anfänger (2)

Fortsetzung von Seite 6

(Reif/Brall)

Die Zeilen 20 - 40 sind uns schon vom ersten Beispiel her bekannt.

In den Zeilen 50 - 70 wird ebenfalls ein Text, gefolgt von einer Variablen und einem weiteren Textstück, ausgegeben. Um dieses zu erreichen, wird in diesem PRINT-Befehl ein Semikolon verwendet.

Semikolon und Komma am Ende einer PRINT-Anweisung

Die Bildschirmposition der nächsten Print-Anweisung wird durch das Zufügen der Zeichen Semikolon oder Komma verändert. Steht am Ende einer Print-Anweisung weder ein Komma noch ein Semikolon, dann wird die Ausgabe der folgenden PRINT-Anweisungen automatisch auf den Anfang der nächsten Bildschirmzeile positioniert. Das bedeutet, daß ohne "," und ";"-Zeichen automatisch ein Wagenrücklauf und Zeilenvorschub ausgeführt wird.

Ein Semikolon bewirkt den lückenlosen Anschluß zur nächsten Ausgabe.
Steht am Ende einer Print-Anweisung ein Semikolon, dann schließt sich die Ausgabe der folgenden PRINT-Anweisung lückenlos an das letzte Ausgabezeichen dieser Print-Anweisung an.
Ein Komma bewirkt einen Sprung zur nächsten Tabulator-Position.
Steht am Ende einer PRINT-Anweisung ein Komma, wird die Ausgabe der folgenden PRINT-Anweisung auf die nächste Bildschirm-Tabulatorposition gesetzt, deren Position entweder auf der Bildschirmmitte der gleichen Zeile oder am Anfang der nächsten Bildschirmzeile liegt.
Werden in einer PRINT-Anweisung mehrere Kommata hintereinander eingegeben, dann springt der Computer so oft auf die nächste Tabulatorposition des Bildschirms, bis alle Kommata abgearbeitet sind.
Wir kehren nun zurück zu unserem Beispiel-Programm und wollen dieses starten!

```
RUN     <RETURN>
BERECHNUNG DER SPANNUNG!
DER STROM BETRÄGT 6 AMPERE
DER WIDERSTAND BETRÄGT
10 OHM
DIE SPANNUNG BETRÄGT
60 VOLT
```

Wollen wir nun die Spannung errechnen, wenn wir den Widerstand auf 70 Ohm setzen, so müssen wir Zeile 30 ändern:

```
30 LET R = 70    <RETURN>
```

Durch Eingeben dieser Zeile wird automatisch die alte Zeile gelöscht.

Wir starten ein zweites Mal:

```
RUN     <RETURN>
BERECHNUNG DER SPANNUNG!
DER STROM BETRÄGT 6 AMPERE
DER WIDERSTAND BETRÄGT
70 OHM
DIE SPANNUNG BETRÄGT
420 VOLT
```

Sicher ist es etwas umständlich, für jede neue Berechnung das Programm umzuschreiben. Um diese Arbeit zu vermeiden, besitzt BASIC den INPUT-Befehl.

Eingabe von numerischen Daten (INPUT)
Oft benötigt man für die Lösung einer oder mehrerer Aufgaben verschiedene Daten. Das Programm verwendet zur Berechnung jedesmal die gleiche Formel mit unterschiedlichen Ausgangswerten.
Die INPUT-Anweisung ermöglicht es uns, Programme zu erstellen, um eine bestimmte Aufgabenstellung mit unterschiedlichen Werten zu lösen. Bauen wir eine solche INPUT-Anweisung in ein Programm ein und starten dieses, dann stoppt das Programm, wenn es die INPUT-Anweisung erreicht hat. Nun kann ein Zahlenwert eingegeben und mit RETURN abgeschlossen werden.

Wir ändern nun unser Beispiel-Programm, indem wir die Zeilen 20 und 30 durch zwei neue INPUT-Zeilen ersetzen:

```
20 INPUT I
30 INPUT R
```

In Zeile 20 wird nun ein Eingabe-Wert verlangt, der der Variablen I zugewiesen wird. In Zeile 30 wird der Wert der Variablen R zugewiesen.

Wir starten unser neues Programm:

```
RUN     <RETURN>
BERECHNUNG DER SPANNUNG!
? 2     <RETURN>
? 40    <RETURN>
DER STROM BETRÄGT 2 AMPERE
DER WIDERSTAND BETRÄGT
40 OHM
DIE SPANNUNG BETRÄGT
80 VOLT
```

Noch einmal mit anderen Werten:

```
RUN     <RETURN>
BERECHNUNG DER SPANNUNG!
? 5     <RETURN>
? 99    <RETURN>
DER STROM BETRÄGT 5 AMPERE
DER WIDERSTAND BETRÄGT
99 OHM
DIE SPANNUNG BETRÄGT
495 VOLT
```

Wir können die INPUT noch etwas übersichtlicher gestalten, indem man einen Text bzw. eine Frage mit dem Kommando kombiniert.

Wir ändern unser Beispielprogramm:

```
20 INPUT „STROMSTÄRKE" ; I
30 INPUT „WIDERSTAND" ; R
```

Nun starten wir noch einmal:

```
RUN                     <RETURN>
BERECHNEN DER SPANNUNG!
STROMSTÄRKE ? 0,5       <RETURN>
WIDERSTAND ? 1000       <RETURN>
DIE STROMSTÄRKE BETRÄGT
0,5 AMPERE
DER WIDERSTAND BETRÄGT
1000 OHM
DIE SPANNUNG BETRÄGT
500 VOLT
```

```
RUN                     <RETURN>
BERECHNUNG DER SPANNUNG!
STROMSTÄRKE ? 0,344
                        <RETURN>
WIDERSTAND ? 500        <RETURN>
DIE STROMSTÄRKE BETRÄGT
0,344 AMPERE
DER WIDERSTAND BETRÄGT
500 OHM
DIE SPANNUNG BETRÄGT
172 VOLT
```

Inzwischen sieht unser Programm wie folgt aus:

```
10 PRINT
„BERECHNUNG DER SPANNUNG!“
20 INPUT „STROMSTÄRKE“ ;I
30 INPUT „WIDERSTAND“ ;R
40 LET U = I * R
50 PRINT
„DIE STROMSTÄRKE BETRÄGT“ ;I; „VOLT“
60 PRINT
„DER WIDERSTAND BETRÄGT“ ;R; „OHM“
70 PRINT
„DIE SPANNUNG BETRÄGT“ ;U; „VOLT“
```

In unserem Beispiel wurden die Zeilen 20 und 30 durch einen „INPUT-TEXT“ ergänzt. Das nach einem Start angezeigte Fragezeichen wird vom Computer selbst ausgegeben, um anzudeuten, daß der Rechner auf eine Eingabe wartet.

Autoren: Elke Reif, Frank Brall

Literatur:

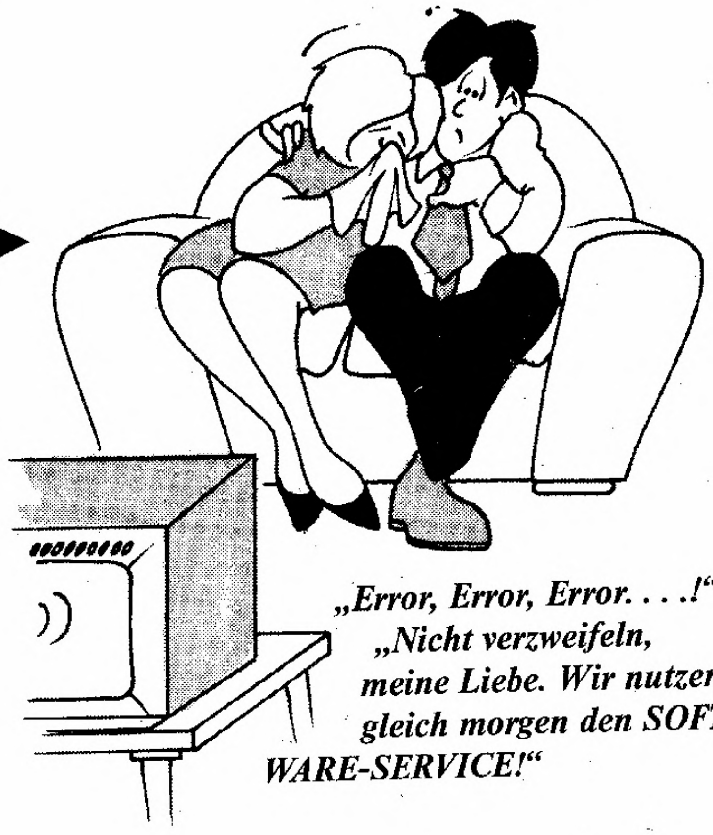
Grundkurs in Basic,
Ulrich Ströbel, Sybex-Verlag
BASIC BREVIER,
Siegmar Wittig, Heinz Heise Verlag
Commodore Handbuch
Spectrum Handbuch
Commodore 116/16/plus/4,
Ekkehard Kaier, Vieweg Verlag

TEIL 3 in der nächsten COMPUTRONIC

Software Service

telefonisch bestellen!
Tel.-Nr. 05651/40693 oder 40643

Wenn Sie sich das Abtippen der Programme ersparen wollen, nutzen Sie doch einfach unseren Software-Service! So wie die Müllers: Abrufkarte (auf S. 45) ausfüllen und abschicken. Oder Bestellung per Telefon! Bei uns werden sämtliche Bestellungen innerhalb nur 1 Woche bearbeitet! Den Service finden Sie auf S. 84.



„Error, Error, Error. . . .!“ „Nicht verzweifeln, meine Liebe. Wir nutzen gleich morgen den SOFTWARE-SERVICE!“

Bitte beachten Sie:

Sie ersparen sich zusätzliche Kosten (bis zu 5,– DM), wenn Sie per Vorkasse (Bargeld, Verrechnungsscheck) bestellen.
Lieferungen ins Ausland erfolgen nur gegen Vorkasse!

Hinweis:

Unser Software-Versand nimmt Reklamationen gern entgegen. Defekte Datenträger werden kostenlos umgetauscht. Beschädigte Ware wird ebenfalls zurückgenommen. Wir bitten unsere Kunden trotzdem, nicht voreilig zu reklamieren. Wir bekommen immer noch angeblich defekte Datenträger zurückgeschickt, die nach einem Test in unserer Computerabteilung jedoch keinerlei Fehler aufweisen. Bitte überprüfen Sie in solchen Fällen (1-3 Reklamationen) Ihre komplette Computeranlage!

Abrufkarte ausfüllen und einsenden an:
Tronic-Verlag, Postfach 41, 3444 Wehretal 1

BASIC-Kurs – Teil 3

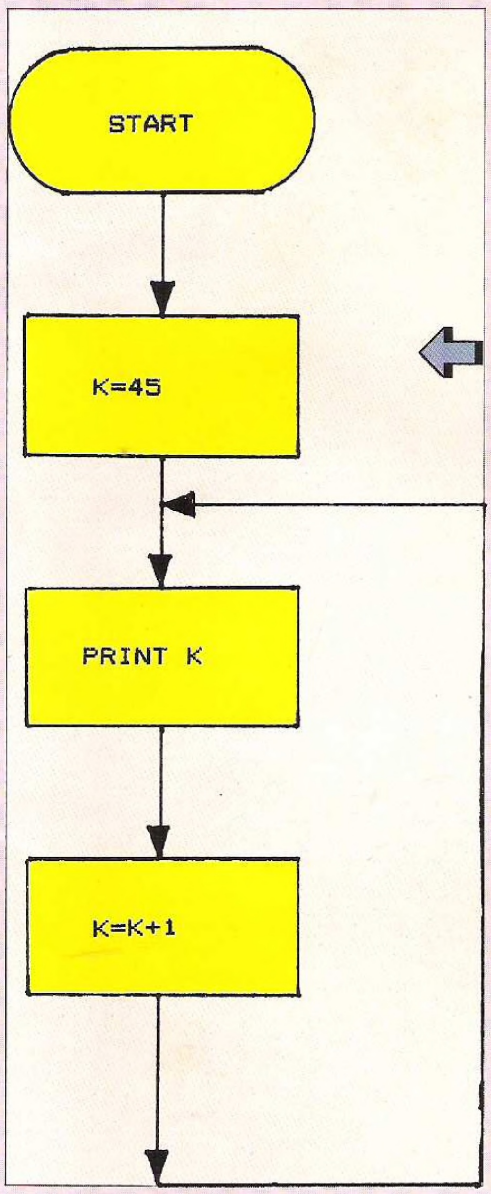
Unbedingte Sprungbefehle

Bis jetzt haben wir gelernt, das Basic-Zeilen fest durchnumeriert sind, und diese in aufsteigender Reihenfolge abgearbeitet werden. Basic besitzt jedoch auch einen „Sprungbefehl", welcher erlaubt, eine bestimmte Zeile anzuspringen. Der Befehl heißt: „GOTO"

GOTO - Befehl

Ein Beispiel demonstriert die Arbeitsweise:

```
10 LET K=45
20 PRINT K
30 LET K=K+1
40 GOTO 20
```



Der Aufbau und die Bedeutung der GOTO-Anweisung ist sehr einfach zu verstehen. Das Schlüsselwort GOTO wird gefolgt von der Nummer der Zeile, die der Computer als nächste ausführen soll. Die Zeilen unseres kleinen Beispielprogramms müßten in folgender Reihenfolge ausgeführt werden.

```
10 --> 20 --> 30 --> 40
<--------------------
10 --> 20 --> 30 --> 40
<--------------------
10 --> ... usw.
```

In unserem Ablaufplan werden die Zeilen 10, 20 und 30 durch ein Kästchen symbolisiert. Die Zeile 40, die GOTO-Anweisung besitzt kein Kästchen, sie wird einfach durch das Einzeichnen entsprechender Pfeile gekennzeichnet.
Was bewirkt eigentlich unser Programm?(!)
Zuerst wird Zeile 10 ausgeführt und der Variablen K der Wert 45 zugeordnet.
In Zeile 20 wird dieser Wert der Variablen K auf dem Bildschirm ausgegeben.
In Zeile 30 wird der Inhalt der Variablen K um 1 erhöht.
In der letzten Zeile wird mit dem GOTO-Befehl auf die Zeile 10 zurückgesprungen.
Nach dem Start des Programms durch RUN werden also die Werte ab der Zahl 45 auf dem Schirm ausgegeben.

```
45
46
47
48
49
50
51
bis unendlich
```

Wie zu erkennen ist stellt unser kleines Programm eine „endlose" Schleife dar, d. h. es wird immer wieder mit wachsenden Werten von Y durchlaufen. Was uns sinnvoll erscheint ist eine Methode, mit deren Hilfe wir dem Computer angeben können, wie oft er eine Schleife durchlaufen soll.

Die bedingte Verzweigung - IF ... THEN - Anweisung

Die IF ... THEN Anweisung ermöglicht es innerhalb eines Basic-Programmes Entscheidungen zu treffen. Diese Anweisung besteht aus den beiden Teilen:

IF (logischer Ausdruck) THEN (BEFEHL)

Wenn der Computer eine If-Anweisung ausführt, wird der logische Ausdruck berechnet und festgestellt, ob dieser Wert wahr oder unwahr ist.

1. Ist der logische Ausdruck wahr, dann führt der Rechner den hinter THEN angegebenen Befehl aus.
2. Ist der logische Ausdruck unwahr, dann ignoriert der Rechner die Anweisung hinter THEN und führt die nächste Anweisung aus.

Nach dem THEN-Befehl darf jede beliebige Basic-Anweisung folgen:

Beispiele:

```
IF .. THEN LET A=14
IF .. THEN INPUT A$
IF .. THEN PRINT "TEST"
```

Lesen Sie weiter auf S. 75

Der Leser muß entscheiden, wer folgende Preise erhalten soll:

1. Preis: 5000 Mark in bar (Scheck)
2. Preis: Farbmonitor oder 800 Mark in bar
3. Preis: Computer-Zubehör (Beispiel – Wersi-Keyboard) im Gesamtwert von 300 Mark
4. bis 108 Preis: Je ein Abo (wahlweise Computronic, Homecomputer, Compute mit oder CPU).Alle eingesendeten Programm werden berücksichtigt!

Basic-Kurs – Teil Drei

Fortsetzung von Seite 7

Bei den meisten Basic-Interpretern haben folgende Zeilen die gleiche Bedeutung:

```
IF ... THEN GOTO 100
      und
IF ... THEN 100
```

Das bedeutet, wenn dem THEN-Befehl eine Zahl folgt, so wird ein Sprung zu dieser Zeilennummer durchgeführt.

Um in BASIC bedingte Entscheidungen zu programmieren, muß die Möglichkeit existieren, Bedingungen für Gleichheit und Ungleichheit anzugeben. Dies wird durch den Gebrauch der folgenden Vergleichsoperatoren möglich:

gleich	=
kleiner als	<
kleiner oder gleich	<=
ungleich	<>
größer als	>
größer oder gleich	>=

Mit Hilfe der oben aufgeführten Vergleichsoperatoren werden numerische Größen (Zahlen, Variablen oder Ausdrükke) oder Zeichen miteinander zu einer Formulierung verbunden.
Der logische Ausdruck wird auf seinen Wahrheitsgehalt hin überprüft. Ist das Ergebnis wahr, so wird dem logischen Ausdruck der Wert 1, ist es unwahr, der Wert 0 zugewiesen. Aus diesem Grund kann bei Ausdrücken, die nur mit den Zahlenwerten 0 oder 1 arbeiten, der Vergleichsoperator vernachlässigt werden.
Neben den oben aufgeführten Vergleichsoperatoren, kennt Basic auch noch die logischen Operatoren.
Mit den logischen Operatoren AND und OR können in einer IF .. THEN-Anweisung beliebig viele logische Einzelausdrücke miteinander verknüpft werden. beim logischen AND müssen beide Bedingungen, beim logischen Operator OR mindestens eine der beiden Bedingungen erfüllt sein.
Die folgende Tabelle (Wahrheitstabelle) zeigt den Zusammenhang sehr deutlich:

1. Ausdruck	2. Ausdruck
unwahr	unwahr
unwahr	wahr
wahr	unwahr
wahr	wahr
verküpft mit AND	verknüpft mit OR
unwahr	unwahr
unwahr	wahr
unwahr	wahr
wahr	wahr

Eine weitere Verknüpfungsart ist die NOT-Funktion. Zu beachten ist, daß nicht jeder Basic-Interpreter diese Funktion integriert hat. Der logische Operator NOT kehrt den Wert des Ausdrucks, den er modifiziert, um.

(Weiter Seite 76!)

Basic-Kurs – Teil 3

Das bedeutet:
Ist ein logischer Ausdruck wahr, liefert NOT den Wert unwahr.
Ist ein logischer Ausdruck unwahr, liefert NOT den Wert wahr.

Wichtig ist, daß NOT direkt vor dem Ausdruck – der modifiziert werden soll – erscheint.

Die Syntax:
IF NOT (logischer Ausdruck) THEN (Befehl)

Wir wollen nun unsere theoretischen Kenntnisse durch einige Beispiele in die Praxis umsetzen.
IF 4= 9 THEN PRINT "vier gleich neun"
In dieser Verzweigung wird der Wert 4 mit dem Wert 9 verglichen. Da die Werte natürlich ungleich sind, ist die Bedingung nicht erfüllt und die PRINT-Anweisung wird nicht ausgeführt.
Eine solche Verzweigung wird in einem Programm keine Verwendung finden, da die Werte fest in der Verzweigung abgegeben sind und nicht durch das Programm verändert werden können. Wesentlich sinnvoller ist die Verwendung von Variablen!
IF 8<9 THEN PRINT "TRONIC"
Das Wort TRONIC wird auf dem Schirm ausgegeben, da der Wert 8 kleiner, als der Wert 9 ist.
IF 9>2*6 THEN PRINT "SOFTWARE"
Das Wort SOFTWARE wird nicht ausgegeben, da der Wert 9 kleiner ist, als das Produkt aus 2 * 6.
IF O THEN PRINT "HARDWARE"
Diese Anweisung sieht auf den ersten Blick etwas ungewöhnlich aus und bringt so manchen Computer-Freak zum Grübeln.

Der Sinn ist jedoch sehr einfach:
Da kein Vergleich durchgeführt wird, wird der Wert 0, als unwahr interpretiert und somit die PRINT-Anweisung nicht durchgeführt.
IF 64>6 and 6<5 THEN PRINT "BASIC"
In dieser Verzweigung werden zwei Bedingungen, durch die Funktion AND, miteinander verknüpft.

Das bedeutet:
Nur wenn beide Bedingungen erfüllt sind, wird der Basic-Befehl ausgeführt. Dies ist in unserem Beispiel nicht der Fall, da der Wert 6 größer als 5 ist.
IF 64>6 OR 6<5 THEN PRINT "BASIC"
Im Gegensatz zu der obigen Verzweigung werden hier beide Bedingungen durch "OR" verknüpft.

Das bedeutet:
Der Basic-Befehl wird auch dann ausgeführt, wenn nur eine der beiden Bedingungen erfüllt ist. In unserem Beispiel ist dies die erste Bedingung (64 ist größer als 6).

Anwendung der IF .. THEN Anweisung

Wir wollen nun eine Bedingung in unser Beispiel einfügen.

```
10 LET K=45
20 PRINT K
30 LET K=K+1
40 IF K=56 THEN END
50 GOTO 20
```

Wenn wir unser Programm starten, so werden die Zahlen von 45 bis 56 auf dem Schirm ausgegeben. Der Vorgang ist sehr leicht zu verstehen: In Zeile 40 wird in jedem Durchlauf der Wert der Variable K mit dem Wert 56 verglichen. Sind diese Werte identisch, so wird das Programm durch den Befehl END beendet.
Wir können das gleiche auch durch folgendes Beispiel erreichen:

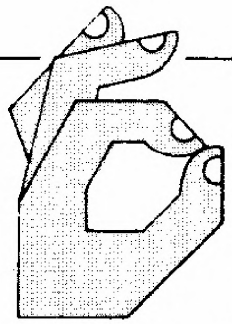
```
10 LET K=45
20 PRINT K
30 LET K=K+1
40 IF K<57 THEN 20
50 END
```

Bei diesem Beispiel wird in Zeile 40 kontrolliert, ob der Wert der Variablen K kleiner als 57 ist, wenn ja, dann wird zur Zeile 20 zurückgesprungen. Im anderen Fall wird die nächste Zeile ausgeführt, und das Programm durch END abgeschlossen.
Wir wollen uns nun an ein etwas komplizierteres Programm heranwagen. Wir wollen ein Programm erstellen, das positive und negative Zahlen erkennt.

```
10 INPUT „ZAHL EINGEBEN";A
20 IF A>=0 THEN 50
30 PRINT „DIE ZAHL IST
   NEGATIV"
40 GOTO 60
50 PRINT "DIE ZAHL IST POSITIV"
60 GOTO 10
```

Wir starten unser Programm durch RUN!

```
ZAHL EINGEBEN ? 4    <RETURN>
DIE ZAHL IST POSITIV
ZAHL EINGEBEN ? –1  <RETURN>
DIE ZAHL IST NEGATIV
ZAHL EINGEBEN ? -999<RETURN>
DIE ZAHL IST NEGATIV
```



```
ZAHL EINGEBEN ? 0    <RETURN>
DIE ZAHL IST POSITIV
```

Der Programmanlauf:
In Zeile 10 wird die eingegebene Zahl der Variablen A zugewiesen.
In Zeile 20 wird kontrolliert, ob der Wert der Variablen A größer oder gleich 0 ist. Wenn ja, so wird nach Zeile 50 gesprungen und die Meldung „DIE ZAHL IST POSITIV" ausgegeben.
Wenn nein, so wird in Zeile 30 die Meldung „DIE ZAHL IST NEGATIV" ausgegeben.
In beiden Fällen wird nach Feststellung der Zahl in die Zeile 10 zurückgesprungen.

UNSER ERSTES SPIEL !!!!!

Wir wollen nun nicht gleich mit einem Action-Game beginnen, sondern als sinnvolle Vorübung ein kleines Ratespiel entwickeln.
Aufgabenstellung: Wir wollen eine Zahl erraten, wobei der Computer uns als Hilfe mitteilt, ob die geratene Zahl zu groß oder zu klein ist. Die Aufgabe besteht darin, mit möglichst wenig Versuchen die Zahl zu erraten. Wie nennen unser Spiel „Ratefix"!

```
10 REM RATEFIX
20 LET E=123
30 INPUT „WELCHE ZAHL ";Z
40 IF Z>E THEN PRINT „Diese Zahl
   ist zu groß !":GOTO 30
50 IF Z<E THEN PRINT „Diese Zahl
   ist zu klein !":GOTO 30
60 PRINT „Richtig geraten !!!"
70 END
```

Bevor wir unser Programm starten, müssen wir die zu erratende Zahl in Zeile 20 eintragen. Am besten läßt man dies von einer anderen Person bewerkstelligen, da es uninteressant ist, wenn man die gesuchte Zahl schon kennt. In unserem Beispiel haben wir die Zahl 123 verwendet.

Wir starten nun mit RUN:

```
Welche Zahl ? 1000        <RETURN>
Diese Zahl ist zu groß !
Welche Zahl ? 100         <RETURN>
```

```
Diese Zahl ist zu klein !
Welche Zahl ? 300          <RETURN>
Diese Zahl ist zu groß !
Welche Zahl ? 150          <HRETURN>
Diese Zahl ist zu groß !
Welche Zahl ? 120          <RETURN>
Diese Zahl ist zu klein !
Welche Zahl ? 125          <RETURN>
Diese Zahl ist zu groß !
Welche Zahl ? 123          <RETURN>
Richtig geraten !!!
```

Wir wollen nun unser Programm noch um einen Zähler, der die Anzahl der Rateversuche, erweitern.

Dazu fügen wir folgende Zeilen ein:

```
21 LET I=0
25 I=I+1
26 PRINT „VERSUCH Nummer: "I
65 PRINT „Anzahl der Versuche:"I
```

In Zeile 40 und 50 ersetzen wir den Befehl GOTO 30 durch ein GOTO 25.

Wir starten nun nocheinmal:

```
VERSUCH Nummer: 1
Welche Zahl ? 500          <RETURN>
Diese Zal ist zu groß !
VERSUCH Nummer: 2
Welche Zahl ? 100          <RETURN>
Diese Zahl ist zu klein !
VERSUCH Nummer: 3
Welche Zahl ? 130          <RETURN>
Diese Zahl ist zu groß !
VERSUCH Nummer: 4
Welche Zahl ? 123          <RETURN>
Richtig geraten !!!
Anzahl der Versuche: 4
```

Wir wollen das Programm nun erweitern, um das lästige Ausdenken einer Zufallszahl vom Computer ausführen zu lassen. Um dies zu bewerkstelligen, benötigen wir zwei neue Basic Befehle: INT und RND

INT (...) - Funktion

INT steht für integer, das englische Wort für „ganze Zahl". INT (...) liefert als Ergebnis „die größte ganze Zahl, die nicht größer als das Argument ist".
Etwas klarer wird dies durch folgende Beispiele:

PRINT INT(2.6) <RETURN>
2

Das Ergebnis ist 2, denn der Wert 2 ist die größte ganze Zahl, die nicht größer als 2.6 ist.

Eine weitere Orientierungshilfe:
Das Ergebnis ist nicht wie eventuell vermutet -2, sondern -3, denn -2 ist größer als das Argument -2.6.
Mit Hilfe einer kleinen Formel lassen sich beliebige Zahlenwerte runden. Das folgende Beispiel rundet Zahlen auf zwei Stellen Nachkommastellen werden abgeschnitten. Dies gilt jedoch nur bei positiven Zahlen.

Ein Beispiel mit negativen Zahlen:

PRINT INT (-2.6)
-3
hinter dem Komma auf bzw. ab:

```
10 INPUT „Zahl ";Z
20 W=INT (W*100+0.5)/100
30 PRINT W
40 GOTO 10
```

Wir starten mit RUN:

```
Zahl ? 12.1234          <RETURN>
12.12
Zahl ? 12.59942         <RETURN>
12.60
Zahl ? 5                <RETURN>
5
```

Der Zufallsgenerator RND

Dieser Befehl ist sicher einer der interessantesten Funktionen Ihres Computers.
Er produziert eine sogenannte Zufallszahl, d. h. eine Zahl, deren Wert nicht vorhersagbar ist (höchstens für Wahrsager).
Das einzige was bekannt ist, ist die Tatsache, daß die erzeugte Zahl zwischen 0 und 0.999 liegt. Dies können Sie kontrollieren indem Sie mehrmals hintereinander den Befehl RND(1) eingeben:

```
PRINT RND(1)            <RETURN>
0.827743801
0.554749226
0.897233831
0.188382000
usw.
```

Eine weitere wichtige Eigenschaft der Funktion RND(...) ist folgende: Ruft man diese Funktion sehr häufig auf, beispielsweise 10000 mal, dann stellt man fest, daß diese 10000 Zufallszahlen gleichmäßig zwischen 0 und 1 verteilt liegen. Das bedeutet: etwa 5000 Zahlen sind kleiner als 0.49999 und 5000 Zahlen sind größer als 0.4999.

Dies ist jedoch nur statistisch und kann in der Praxis natürlich etwas abweichen.
An dieser Stelle sei darauf hingewiesen, daß nicht alle Homecomputer „richtige" Zufallszahlen erzeugen. Einige verwenden in Wirklichkeit eine komplizierte mathematische Formel zur Erzeugung von Pseudo-Zufallszahlen. Wir wollen jedoch nicht näher darauf eingehen, sondern uns ein kleines Beispiel ansehen:

```
10 REM WUERFEL
20 K=INT(6*RND(1))+1
30 PRINT K
40 GOTO 20
```

In diesem Beispiel wird die INT-Funktion mit der RND-Funktion kombiniert.
Die Funktion der Zeile 20 im Einzelnen:
Zuerst wird eine Zufallszahl zwischen 0 und 1 ermittelt.
Diese wird nun mit 6 multipliziert und die Nachkommastellen durch den INT-Befehl abgeschnitten.
Da die Zahlen ab dem Wert 1 beginnen sollen, wird das Ergebnis der Operation noch um 1 erhöht.

Wir starten unser Würfel-Programm:

```
1
6
5
3
3
1
4
3
2
4
```

Wie zu erkennen ist, wird eine zufällige Zahlenfolge zwischen 1 und 6 ausgegeben. Würden wir in unserem Programm die 6 durch eine 49 ersetzen, so könnten wir uns auch die Lottozahlen ausgeben lassen.

So nun zurück zu unserem RATEFIX

Wir sind nun in der Lage das Ratefix Programm zu perfektionieren. Wir wählen die Zahlenwerte so, daß immer eine Zufallszahl zwischen 1 und 1000 ermittelt wird:

```
10 REM RATEFIX
20 PRINT „Ein neues Spiel !!!"
30 E=INT(1000*RND(1))+1
40 I=0
50 I=I+1
60 PRINT „Versuch Nummer:"I
```

```
70 INPUT „Welche Zahl ";Z
80 IF Z>E THEN PRINT „DIESE
   ZAHL IST ZU GROSS!":GOTO 50
90 IF Z<E THEN PRINT „DIESE
   ZAHL IST ZU KLEIN!":GOTO 50
100 PRINT „RICHTIG GERATEN !!!"
110 PRINT „ANZAHL DER
   VERSUCHE:"I
120 GOTO 20
```

Unser erstes Programm ist erstellt !!!

Autor: Frank Brall/Tronic-Verlag

Literatur: Grundkurs in Basic, Ulrich Stroebel, Sybex-Verlag
Basic Brevier, Siegmar Wittig, Heinz Heise Verlag
Commodore Handbuch
Spectrum Handbuch
Commodore 116/16/plus4, Ekkehard Kaier, Vieweg Verlag

BASIC-KURS-TEIL 4 in der nächsten Computronic

Tips & Tricks Nützliche HILFEN

Auch in diesem Heft bieten wir wieder „Hilfen" für den geplagten „Software-Player" an. In der aktuellen Ausgabe sind wieder „Manipulier-Tricks" für Apple II-Spiele zu finden. Desweiteren kommen auch die „64er-Freunde" auf ihre Kosten: Wie die Redaktion meint, ist einiges Nützliches dabei. Doch: urteilen Sie selbst! (M. K.)

APPLE II

Bug Battle:

call-151
3feg:ea ea ea
93ig: startet das Spiel

Rear guard:

drücken Sie (crtl - T)
dann den Level

Teleport:

(reset)
4idi:ea ea ea
5f8cg : startet das Spiel

Snack Attack:

call-151
5b28: # der Leben
6fdg : startet das Spiel

Phaser Fire:

call-151
452e: # der Schiffe
3ffdg : startet das Spiel

Mehr Leben bei **JAWBREAKER**
run Jawbreaker
reset
84b: # der Leben
800g : startet das Spiel

Hilfe für **Apple Panic:**
call-151
768:ea ea ea
4000g : startet das Spiel

Mehr Leben bei **Nightmare Gallery**
call-151
1361: # der Leben
600g : startet das Spiel

Mehr Leben bei **Canyon Climber:**
call-151
3000: # der Leben
300g : startet das Spiel

TI-99

1. Folgendes Programm ermöglicht es, ein Programm, das irrtümlich in X-Basic geladen wurde und nur in TI-Basic läuft, starten zu lassen ohne es neu zu laden.

```
1 !@P-
2 Call Char
```

Danach wird RUN eingegeben und die Zeilen 1 und 2 gelöscht.

2. Mit diesem Befehl läßt sich die QUIT-Taste sperren

```
Call Init
Call Load(-31896,16)
```

Die Sperrung wird aufgehoben durch:

```
Call Load(-31896,0)
```

3. Der beim INPUT-Befehl zu hörende Ton, kann durch nachstehenden Befehl unterdrückt werden.

```
10 Call Sound(500,880,30)
20 Input A$
```

Der Grund dafür liegt in Zeile 10.

Spielertips für den Commodore 64

Aligata Blagger	- POKE 53264,126
Bagitman	- POKE 19013,189
Bat Attack	- POKE 11061,234
Battle through Time	- POKE 22045,255
Battle Zone	- POKE 8909,100
Black Hawk	- POKE 8290,255
Bruce Lee	- POKE 5686,128
Cavelon	- POKE 23789,255
China Miner	- POKE 34623,234
Crisis Mountain	- POKE 2665,238
Dare Devil Dennis	- POKE 29173,255
Dig Dug	- POKE 10473,255
Dinkey Doo	- POKE 12296,165
Donkey Kong	- POKE 12118,234
Falcon Patrol	- POKE 16764,234
Fort Apocalypse	- POKE 36364,234
Galaxy	- POKE 3369,230
Hard Hat Mack	- POKE 16877,173
H.E.R.O.	- POKE 14652,25
High Noon	- POKE 18033,255
Hunchback	- POKE 5704,50
Jumpman Junior	- POKE 9450,173
Jungle Hunt	- POKE 2242,165
Kickman	- POKE 7424,230

Atari-/Commodore-Joysticks am TI 99/4A

Da die Handlichkeit der Texas Joysticks für Spiele häufig ungenügend ist, hat sich der Autor mit der Pinbelegung auseinandergesetzt!

Der Quick Shot II

Diejenigen, die den Joystick „Quick Shot II" benutzen wollen, benötigen eine weitere Lötung im Joystick, um den „Auto fire" funktionsfähig zu bekommen.
Dazu werden die vier Kreuzschlitzschrauben an der Unterseite des Joystick herausgedreht und das Unterteil abgeklappt (Vorsicht, keine Kabel abreißen).
Nun muß eine Verbindung vom leichtgedrehten Haltestift des Schalters zu dem Pin mit dem roten Kabel an der Unterseite des Schalters gelötet werden.
Ohne diese Lötung funktioniert der Aktionsknopf bei Stellung des Auto fire auf „on" nicht.

Pin - Belegung bei TI 99/4A für Joysticks

Die Pinnummern wurden vom eingesteckten Stecker abgeschrieben.

Computereingang		Wert	Joyst. Atari/Commodore	
Pin	Bedeutung	ein / aus	Pin	Bedeutung
2	Ausgang Joyst. 2		1	– Höhenwert Ausgang
3	+ Höhenwert Eing.	+ 4 / 0	2	+ Höhenwert Ausgang
4	Keywert Eing.	+18 / –1	3	– Seitenwert Ausgang
5	– Seitenwert Eing.	– 4 / 0	4	– Seitenwert Ausgang
7	Ausgang Joyst. 1		6	Keywert Ausgang
8	– Höhenwert Eing.	– 4 / 0	8	Computer Eingang
9	+ Seitenwert Eing.	+ 4 / 0		

Anschließend wird das Unterteil wieder angeklappt und die vier Schrauben fest angedreht.
Für das Anschließen von zwei Joysticks benötigt man ein siebenadriges Flachbandkabel, zwei neunpolige Joystickstecker und eine neunpolige Joystickbuchse.
Das Kabel nicht zu kurz kaufen, lieber 10 cm zu lang als 1 cm zu kurz. 30 cm sollten aber reichen. Das Kabel kann auch, falls länger als 1 Meter als Verlängerung dienen.

(Jürgen Böhning)

Tips und Tricks für den VC-20

A. ON ERROR GOTO ...

Folgende Einzeiler simuliert den Befehl unter Programmkontrolle:

```
A$="90"+stR(nnnnn)+cH(13):L=LEN
(A$):P0198,L:f0A=1TOL:P0630+A,ASC
(mI(A$,A)):nE
```

nnnnn ist die Zeilennummer der Errorbehandlungsroutine. Die etwas unorthodoxe Schreibweise der Befehle soll die Kurzformen darstellen; mit den Kurzformen umfaßt die Zeile insgesamt 77 Zeichen, es bleibt also noch Platz für eine fünfstellige Programmzeilennummer.

B. UNNEW.

```
10 FOR A=525 TO 577:READ d:POKE
   A,D:NEXT
20 POKE43,167:POKE44,2:POKE45,220:
   POKE46,2:CLR:SAVE "UNNEW",1,1
30 DATA 160,  3,200,177, 43,208,251,200
   200,152
40 DATA 160,  0,145, 43,165, 44,200,145,
   43,133
50 DATA 60,160,  0,132, 59,162,  0,200,
   208,  2
60 DATA 230, 60,177, 59,208,245,232,
   224,  3,208
70 DATA 242,200,208,  2,230, 60,132,
   45,164, 60
80 DATA 132, 46, 96
```

Der Einsatz erfolgt mit SYS 525:CLR
Das Programm ist an einem von dem Überschreiben halbwegs geschützten Platz im Stapelspeicher untergebracht und somit stets – auch nach einem Reset – verfügbar.

C. Verlangsamter PRINT

Folgende Zeile ermöglicht die Bestimmung der Druckgeschwindigkeit auf dem Bildschirm. Als Übergabeparameter ist ein vorher definierter String, hier A$, erforderlich. Die Geschwindigkeit wird durch die B-Schleife bestimmt:

```
FOR A = 1 TO LEN(A$):PRINT
                MID$(A$,A1);:
FOR B = 1 TO 22: NEXT B,A
```

D. Wieder: Tastaturverriegelung

Daß man die Tastatur mit POKE 649,0 verriegeln kann, ist längst bekannt: dieselbe Wirkung erzielt man auch mit:

```
POKE 37155,255
```

E. INSTRING

Folgende Zeile simuliert die INSTRING-Funktion: als Übergabeparameter müssen zwei String vorher definiert werden:
A$ ist der Suchstring
B$ ist der zu durchsuchende String
Die Zeile läßt sich als einzeiliges Unterprogramm abrufen, wenn die entsprechenden Befehle (s. Schreibweise!) in Kurzform eingegeben werden:

```
A=LEN(A$):B=LEN(B$):N=0:fOI=
1toB-A+1:IFA$()mI(B$mI,A)THENnEI:
N=I:I=B-A+1:nEI
```

In dieser Form bleibt noch Platz für eine fünfstellige Programmzeilennummer sowie reT [RETURN-Kuerzel]

F. Zentrierter Bildschirmausdruck

Als Übergabeparameter ist ein String für jede Bildschirmzeile erforderlich, z. B. hier A$:

```
PRINT TAB(( 22 - LEN(A$) ) / 2)A$
```

G. PRINTE Unterprogramm mit einem Parameter

Die inzwischen wohl weltbekannte Methode PRINTE [PRINT AT] zu simulieren:

```
POKE781,X:POKE782,Y:POKE783,0:
SYS65520:PRINT"string"
```

läßt sich zu einem Unterprogramm umwandeln, das nur gedruckt werden soll.

```
 10 P=255:GOSUB1000:PRINT"string"
1000 POKE781,p/22:POKE782,P-22*
     PEEK781:POKE783,0:
     SYS65520:RETURN
```

Wer nicht auf Anhieb den Wert für P ausrechnen kann, hat die Möglichkeit dieses dem Computer zu überlassen, und zwar mit der Zeile:

P = (Y*22)+X

H. Selbstmodifizierendes Programm

Wenn man die Speicherzellen kennt, in denen die tokenisierten BASIC Befehle stehen, kann man durch hineinPOKEn von anderen Tokenwerten den Programmablauf ändern, z. B.:

0 REM 200

Die Zeilen 1 bis 198 stehen für Programmphase 1 zur Verfügung

```
199 EQ=4+PEEK(43)+256*PEEK(44):
POKE EQ,137:END
```

Ab Zeile 200 läuft dann Programmphase 2. Durch Zeile 199 ist der Tokenwert für GOTO jetzt wirksam. Zeile 0 sieht beim Listing dann so aus:

0 GOTO 200

Dieses einfache Beispiel soll andere User anregen, mit den dadurch gebotenen Möglichkeiten zu experimentieren.

I. Tastaturbieper

Für diejenigen, die beim Eintippen eine akustische Bestätigung bevorzugen:

```
10 FOR A = 678 TO 701:READ D:POKE
   A,D:NEXT
20 DATA 169, 15,141, 14,144,120,169,
        78,141, 20
30 DATA   3,169,  3,141, 21,  3, 88,
        96,165,197
40 DATA 201,128,240,  7,101,197,105,
        128,141, 12
50 DATA 133, 76,191,234
```

Mit SYS 678 darf gebiept werden.

J. Blinkende Bildschirmzeile

Gelegentlich möchte man einen Ausdruck am Bildschirm von der Umgebung abheben: eine Möglichkeit besteht darin, den PRINT String an - und ausblinken zu lassen. Dies kann folgendermaßen geschehen:

```
 10 PO$="crsrhome+22*crsrdown"
 20 ME$="das,was blinken soll"
 30 Y= [bildschirmzeile]:X= [Spalte]
 40 NR= [anzahl der blinkphasen]:
    GOSUB1000
1000 BL=0
1020 FOR I= 1 TO NR
1030 PRINT LEFT$(PO$,Y+1);TAB(X);
     CHR$(18+ABS(128*BL));ME$
1040 TT=TI+J1
1050 BL=BL=0
1060 IF TI < TT THEN1070
1070 NEXT
1080 RETURN
```

TI in Zeile 1040 und 1060 ist die Systemvariable für die Systemuhr.

K. GOTO berechnet

Mit folgenden Einzeiler kann man zu einer vorher berechneten Zeile springen; Voraussetzung dafür ist, daß die Variable LN den Wert der Zeile enthält, bevor man zu dieser Zeile springt - P0785,188:P0786,200: POLN,PE(LN):LN=USR(0)

★☆★☆★☆★☆★☆★☆★☆★☆★

Basic-Kurs - Teil 5

Stringverarbeitung

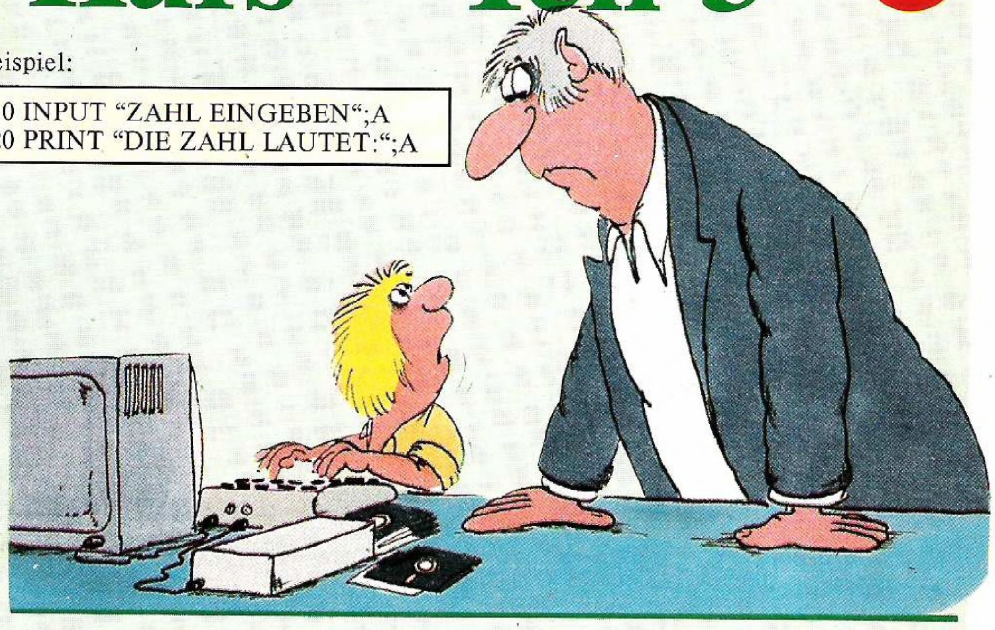
In den letzten beiden Teilen haben wir sehr viele neue Funktionen kennengelernt. Alle diese Funktionen haben gemeinsam, daß sie nur Zahlen verarbeiten können. Nun ist allgemein bekannt, daß Computer auf der Tastatur auch Buchstaben besitzen. Wir wollen uns nun mit den Funktionen befassen, die auch Buchstaben und andere Sonderzeichen verarbeiten können, man spricht in diesem Falle von Funktionen der Stringverarbeitung.

Was ist ein String?

Im ersten Teil unseres Kursus haben wir schon aufgezeigt, welche Zeichen der Computer darstellen und verarbeiten kann. Ein String ist eine Aneinanderreihung einer bestimmten Anzahl von beliebigen Zeichen. Ein Beispiel war unser kleines Mini-Programm im Kurs-Teil 2. Dort verwendeten wir einen String in einer PRINT- sowie in einer INPUT-Zeile.

Beispiel:

```
10 INPUT "ZAHL EINGEBEN";A
20 PRINT "DIE ZAHL LAUTET:";A
```



STRING ist das englische Wort für Kette und bedeutet in diesem Fall eine Art Zeichenkette.

Zum besseren Verständnis könnte man das Wort STRING durch das Wort Text, und das Wort Stringverarbeitung durch das Wort Textverarbeitung ersetzen.

Lesen Sie weiter auf der nächsten S.

Dieser Kurs läßt Sie zum perfekten Basic-Programmierer werden!

Von „Textverarbeitung“ kann jedoch in dem obigen Beispiel noch keine Rede sein. Die Texte werden lediglich genauso, wie sie in der Programmanweisung vom Programmierer formuliert wurden, auch wieder auf dem Bildschirm ausgegeben (natürlich auch vorhandene Rechtschreibfehler).
Nun hat ein Computer natürlich noch etwas mehr zu bieten als INPUT und PRINT, als erstes wären hier die Stringvariablen zu nenen. Was man unter einer Variable versteht haben wir schon einmal angesprochen, jedoch kennen wir noch keine String - bzw. Textvariablen. Eine solche Variable unterscheidet sich von der numerischen Variable dadurch, daß dieser keine Zahl sondern ein ganzer String zugeordnet wird. Um dem Computer mitzuteilen wann es sich um eine numerische und wann es sich um eine Stringvariable handelt, muß der Variablenname entsprechend gekennzeichnet werden. Es ist üblich die Variablennamen, hinter denen sich Texte verbergen, mit einem angehängten $ (Dollar)-Zeichen von den Namen für numerische Variablen zu unterscheiden. Folgenden Textvariablen wären denkbar:

```
A$,B$,C$,T1$
```

Die meisten Computer erlauben auch längere Variablennamen wie:

```
PUNKTE$,OTTI$,NAME$
```

Denken Sie bei längeren Variablennamen daran, daß die einzelnen Computer eine unterschiedliche Anzahl von Zeichen interpretieren. Dadurch ist es möglich, daß die Variablennamen PUNKT$ und PUDEL$ als gleich interpretiert werden.
Wie bei den numerischen Variablen werden die Stringvariablen durch eine Zuweisung definiert. Im Gegensatz zu den numerischen Variablen kann hier der LET-Befehl entfallen.
Einige Beispiele:

```
10 A$= "OTTI GEHT NACH HAUS“
20 B1$="NAME:“
30 NAME$="FRANK“
40 PRINT A$
50 PRINT B1$
60 PRINT NAME$
```

Wir starten unser Programm:

```
RUN <RETURN>
OTTI GEHT NACH HAUS
NAME:
FRANK
```

Wie in dem Beispiel zu erkennen ist, muß bei der Zuweisung einer Stringvariable der eigentliche Text in Anführungszeichen gesetzt werden. Der Begriff Text kann recht locker gefaßt werden. Zu Recht spricht man nicht von „Buchstaben“, sondern von „Zeichen“. Dies soll verdeutlichen, daß man in einem String auch Sonderzeichen sowie Ziffern unterbringen darf:

```
10 A$="12345“
20 B$="$100 + $300“
30 C$=“  “
```

Wie die Zeilen 20 und 30 unseres Beispieles zeigen, können auch Leerzeichen in einem String festgehalten werden. Viele Computer besitzen auch Grafikzeichen wie Figuren, Herzen oder Linien, auch solche Zeichen können in einem String verwendet werden. Es gibt jedoch einige Zeichen, die nicht verwendet werden können:

```
10 T$="ER SAGTE “HALLO“ UND
GING“
```

Die meisten Computer würden bei dieser Zeile annehmen, daß die Zeichenkette bei dem Wort „ER“ beginnt und vor dem Wort „HALLO“ endet. Dadurch können je nach Computertyp unterschiedliche Reaktionen hervorgerufen werden, entweder meldet sich der Computer mit einer Fehlermeldung oder er interpretiert die Zeile fehlerhaft und fährt im Programm weiter. Man sollte sich also immer erst im Handbuch seines Computers darüber informieren, wie dieser es gerne haben möchte. Im Zweifelsfalle sollte man durch eine kleine Testzeile (beispielsweise wie oben) feststellen wie der Computer auf das Anführungszeichen reagiert.

Wie lang darf ein String sein?

Wie wir in unseren Beispielzeilen schon gesehen haben, können in Stringvariablen Texte unterschiedlicher Länge zugewiesen werden. Die Programmiersprache Basic bietet dem User tatsächlich diese Möglichkeit. Je nach Computertyp und Basicversion ist eine maximale Stringlänge vorgegeben. Bevor Sie längere Strings verwenden, sollten Sie die Länge im Handbuch nochmals nachlesen.
Hier eine kleine Aufzählung einiger Computer:

Computertyp	Stringlänge
C64	255
VC20	255
C16	255
CPC 464	255
SPECTRUM	unbegrenzt
ZX81	unbegrenzt
MSX	100 (definierbar)

Die Stringlänge läßt sich leicht durch eine Basic-Zeile feststellen:

```
A$="“:FOR I=1 TO 255:A$=A$+
“A“:NEXT I
```

Gibt Ihr Computer nach dem Start dieser Zeile keine Fehlermeldung aus, so kann Ihr Computer mindestens 255 Zeichen in einem String ablegen. Durch Erhöhen oder Erniedrigen der Zahl 255 können Sie schnell die maximale Stringlänge ermitteln.
Wird die maximale Stringlänge überschritten so melden sich die meisten Computer mit der Meldung “?STRING TOO LONG ERROR“
Einige Computer ignorieren auch einfach die überschüssigen Zeichen, ohne dies durch eine Fehlermeldung anzuzeigen. Dies stellt eine gefährliche Programmfalle dar und hat schon manchen zur Verzweiflung gebracht.

Handhabung von Strings

Ist einer Variable einmal ein String zugeordnet worden, so kann diese jederzeit auch wieder neu definiert werden:

```
10 A$="SCHMIDT“
20 PRINT A$
30 A$="MEIER“
40 PRINT A$
40 A$="“
60 PRINT A$
```

Wir starten unser Programm:

```
RUN <RETURN>
SCHMIDT
MEIER
```

In diesem Beispiel wird die Variable A$ drei mal definiert, wobei sie in Zeile 50 gelöscht wird.
Es gibt noch eine andere Methode, einer Textvariablen einen Wert zuzuweisen, und zwar mittels unserer INPUT-Anweisung. Folgendes Beispiel demonstriert dies:

```
10 REM *** STRING-ZUWEISUNG ***
20 INPUT A$
30 PRINT A$
40 GOTO 20
```

Ein anderes Beispiel:

```
10 REM ** STRING-ZUWEISUNG **
20 A$="NAME
30 B$="STRASSE"
40 C$="ORT"
50 INPUT A$,A1$
60 INPUT B$,B1$
70 INPUT C$,C1$
80 PRINT A$"="A1$
90 PRINT B$"="B1„
100 PRINT C$„=„C1$
```

Testen Sie selbst die Funktion dieser Programme und versuchen Sie ein eigenes kleines „STRING-Programm" zu erarbeiten.
Im nächsten Teil gehen wir auf die ersten Stringfunktionen ein und schreiben ein kleines Programm.

Autor: Frank Brall

Literaturquellen:
Grundkurs in Basic, Ulrich Stroebel, Sybex-Verlag
Basic Brevier, Siegmar Wittig. Heinz Heise Verlag
Commodore Handbuch
Spectrum Handbuch
Commodore 116/16/plus4, Ekkehard Kaier, Vieweg Verlag

BASIC-KURS-TEIL 6 folgt in der nächsten Computronic

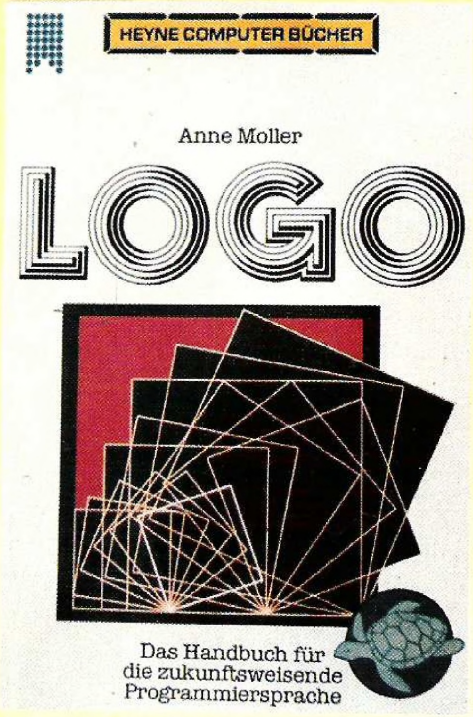
Computronic-Bücher-Kiste

Das Handbuch für die zukunftsweisende Programmiersprache

LOGO
Anne Moller

Logo, eine höhere Programmiersprache, wurde von Professor Seymour Papert in den Sechziger Jahren als Programmiersprache für Kinder erfunden. Nun, da diese Sprache auch in Deutschland auf dem Vormarsch ist, erschien bei Heyne zur rechten Zeit Anne Mollers Buch mit dem naheliegenden Titel „LOGO".

Die Autorin vermittelt in ihrem Werk dem Leser einen umfassenden Einblick in Arbeitsweise und Möglichkeiten der Sprache Logo. Diese einfache und dennoch vielfältige Sprache hat es sich zur Aufgabe gestellt, Computer-Logik sichtbar zu machen. Mittels eines Lichtpunktes (der - bei einigen Computer-Systemen stilisierten - sogenannten „Schildkröte") können auf dem Monitor durch einfache Wortbefehle grafische Gebilde gezeichnet werden. Darüberhinaus sollen Kinder mit dieser Sprache ein besseres Verständnis für Mathematik und Geometrie bekommen. Neben der Programmierung von beweglichen Bildern werden in „LOGO" das gesamte Logo-Vo-



kabular und seine Dialekte vorgestellt. Auch die Themengebiete „Logo im Unterricht" und „Logo in der Informatik" werden ausführlich erörtert.
Da die Programmiersprache Logo bereits in vielen Schulen angewandt wird, ist „LOGO" von Anne Moller sicher ein nützlicher Ratgeber für fast jeden Computer-Anwender. Und noch ein Hinweis: Die Sprache Logo wird inzwischen von den meisten Heimcomputer-Herstellern angeboten; dazu gehören unter anderem IBM, Commodore, Atari, Tandy, Sinclair, Apple und viele andere.

„LOGO" ist in „Heyne Computer-Buch" (Wilhelm Heyne Verlag, München) erschienen. ISBN 3-453-47063-X, Preis 14,80 DM. (bez.)

Lernen mit dem Familien-Computer
Sabine Quinten-Eirich

„Lernen mit dem Familien-Computer" richtet sich an Eltern und Erzieher auf der einen und an Kinder im Alter von fünf bis zwölf Jahren auf der anderen Seite (welche bei der Beschäftigung mit dem Buch jedoch auf die Unterstützung durch Erwachsene angewiesen sein werden).

Die Autorin, Sabine Quinten-Eirich, studierte Sozialpädagogik in München und nutzt und testet seit drei Jahren (als Mutter zweier Kinder sicher in sehr kompetenter Weise) Computer-Lehr- und -Lernprogramme. Neben einer umfangreichen Übersicht über das Angebot an Lehr- und Lernprogrammen gibt sie dem Leser die Möglichkeit, sich über Zeichen-, Musik- und Sprachlernprogramme für Kinder zu informieren.
Ihr Hauptaugenmerk legt Sabine Quinten-Eirich hierbei auf die Frage, ob überhaupt, wann und unter welchen Voraussetzungen Kinder an den Computer herangeführt werden sollten. Diese Problemstellung schlängelt sich gewissermaßen als „Roter Faden" durch das gesamte Werk.

„Lernen mit dem Familien-Computer" ist erschienen in der Reihe „Heyne Computer-Bücher", Wilhelm Heyne Verlag, München. ISBN 3-453-47053-2. Preis 12,80 DM. (bez.)

Basic Kurs Teil 6

Im letzten Teil unseres Kurses haben wir schon einiges über Strings gelernt. Wir werden diesmal sehen, daß sich mit Strings noch vielmehr bewirken läßt.

STRINGFUNKTIONEN

Oft kommt es vor, daß man im Verlauf eines Programmes die Anzahl der Zeichen einer Zeichenkette wissen muß. In Basic dient zu diesem Zweck die Funktion LEN. Angenommen wir haben die Variable NAME$ mit dem String "FRANK" definiert. Wollen wir nun die Anzahl der Buchstaben wissen, so geben wir den folgenden Befehl ein:

```
PRINT LEN (NAME$)
```

Nach Betätigen der RETURN-Taste erscheint die Zahl 5. Dies bedeutet, daß die Variable NAME$ mit einem 5 Zeichen langen String belegt ist. Selbstverständlich läßt sich diese Zahl auch einer Variablen zuordnen:

```
N=LEN(NAME$
```

Ebenfalls ist es möglich, als Argument einen String direkt in Klammern zu setzen, ohne das eine Variablenbezeichnung benötigt wird:

```
N=LEN("FRANK")
```

Der LEN-Befehl kann verschiedene Aufgaben in Programmen durchführen. Durch die Verbindung mit einer IF-THEN Bedingung kann er beispielsweise als Verzweigung dienen.

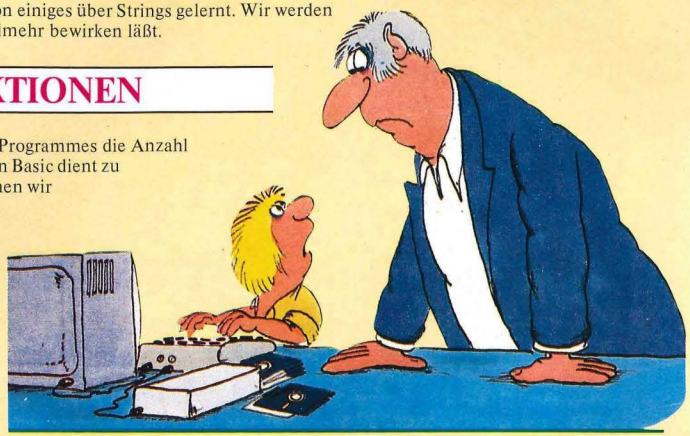
```
IF LEN (NAME$)=4 THEN 100
```

Eine andere Möglichkeit besteht darin, das Ergebnis dieser Funktion in einem Ausdruck weiterzuverarbeiten.

```
A=2*LEN(NAME$)
```

Das Argument des LEN-Befehles kann ebenfalls aus einem Ausdruck bestehen. Das folgende Beispiel zeigt, daß auch Leerzeichen als Zeichen mitgezählt werden.

```
10 A$="kleiner"
20 B$=" "
30 C$="OTTI"
40 E=LEN(A$+B$+C$)
50 PRINT A$+B$+C$
```



Mensch Vati, Du immer mit den „blöden" Hausaufgaben, der Basic-Kurs in Computronic ist doch viel wichtiger!

Wir starten unser Beispiel:

```
RUN <RETURN>
KLEINER OTTI
12
```

Zu beachten ist, daß nicht alle Basic-Versionen das Verbindungszeichen "+" zum Verketten von Strings erlauben.

LEFT$ - String von links

Die Funktion LEFT$, die 2 Parameter besitzt, löst den linken Teil aus einer gegebenen Zeichenkette heraus. Der erste Parameter besteht aus der Zeichenkette, aus der von links her so viele Zeichen herauszulösen sind, wie der zweite Parameter angibt. Dieser kann Werte zwischen 0 und 255 annehmen. Ist er größer, so wird die Fehlermeldung Illegal Quantity Error ausgegeben.

Wir wollen die Funktion von LEFT$ an einem kleinen Beispiel demonstrieren:

```
10 NAME$="TRONIC-VERLAG"
20 TEIL$=LEFT$(NAME$,6)
30 PRINT NAME$
40 PRINT TEIL$
```

Wir starten unser Beispiel:

```
RUN <RETURN>
TRONIC-VERLAG
TRONIC
```

Nimmt man als zweites Parameter statt dem Wert 6 den Wert 0, so wird nur ein Leerstring übergeben. Ist der zweite Parameter größer als der erste, so werden alle Zeichen übermittelt, ohne daß eine Fehlermeldung zu befürchten ist.
Besitzt der Computer kein Locate-Befehl oder eine andere Möglichkeit zum Positionieren des Cursors, so kann der LEFT$-Befehl leicht diese Aufgabe übernehmen. Bedingung ist, daß der Computer über Kontrollzeichen den Cursor bewegen kann. Ist dies der Fall, so werden zwei Strings angelegt, wobei der erste Zeichen enthält, die den Cursor nach rechts bewegen, und der zweite Zeichen, die den Cursor nach unten lenken. Durch die Verwendung zweier LEFT$-Befehle kann somit der Cursor an jede beliebige Stelle gesetzt werden. Zur besseren Handhabung empfiehlt es sich, ein geeignetes Unterprogramm zu schreiben, das nur mit GOSUB angesprungen wird.
Das nachfolgende Beispiel demonstriert

dies bei einem Commodore-Computer. Nachdem die Variablen X mit der horizontalen Koordinate und die Variable Y mit der vertikalten Variable belegt wurde, wird das Unterprogramm ab Zeile 50000 angesprungen und somit der Cursor gesetzt. Zu beachten ist, daß X- und Y-Koordinaten ab 0 gezählt werden.

```
1 PRINT"▽"
10 Y=3:X=8:GOSUB 50000
20 PRINT "TEST"
30 END
50000 REM UNTERPROGRAMM ZUM CURSOR SETZEN
50010 REM VARIABLEN:
50020 REM X GLEICH HORIZONTALE POSITION
50030 REM Y GLEICH VERTIKALE POSITION
50040 REM
50050 PRINT "▽";:REM CURSOR IN LINKE OBERE ECKE (AUSGANGSPUNKT)
50060 X1$="▷▷▷
```

BASIC-KURS Teil 7

Nachdem wir in der letzten Computronic-Ausgabe mit dem Kapitel Stringverarbeitung begonnen haben, sind wir jetzt in der Lage erste nützliche Programme zu entwickeln. In der 7. Folge unseres Basic-Kurses stellen wir nun weitere wichtige Stringfunktionen vor.

Konvertierungs- und Zeichenkettenfunktionen

Wie wir schon in den ersten beiden Teilen unseres Kurses gelernt haben, entspricht jedes dargestellte Zeichen, sowie auch alle Buchstaben und Zahlen, einem speziellen Code: Dem ASCII-Code. Da es bei speziellen Anwendungen von Vorteil ist, statt mit dem Zeichen, mit dem Code zu arbeiten, kennt der Basic-Interpreter zwei Umwandlungs-Befehle. Dadurch ist es beispielsweise relativ einfach, Namen nach dem Anfangsbuchstaben zu sortieren, da den Buchstaben im ASCII-Code aufsteigende Werte zugeordnet sind.

ASC (...)
Die ASC-Funktion ermöglicht es, den internen Zahlenwert eines Buchstabens oder eines anderen Zeichen's zu ermitteln. Tippen wir einmal das folgende Beispiel in unseren Rechner:

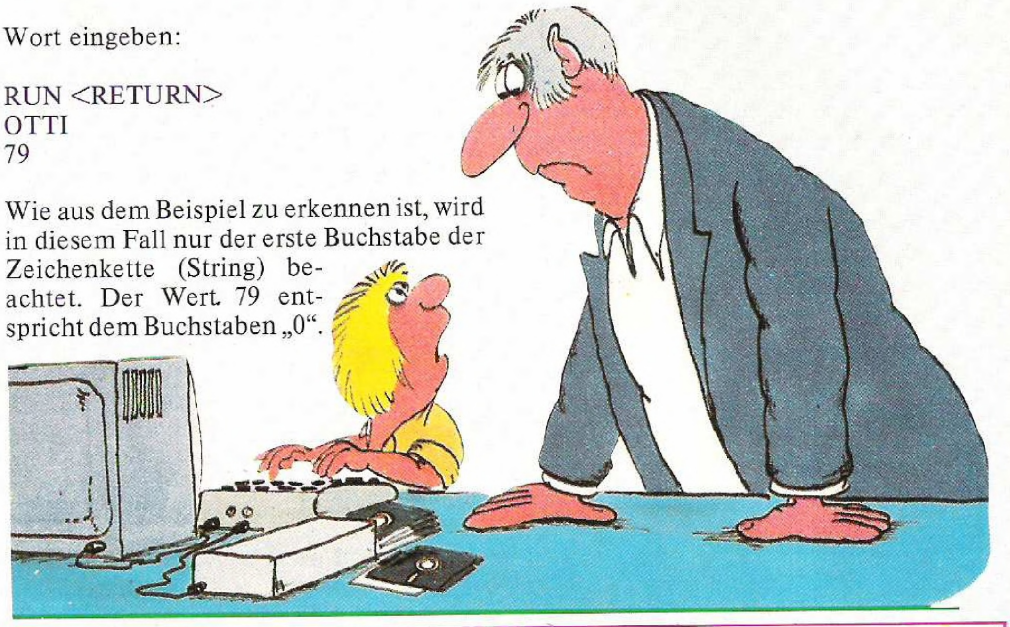
```
10 INPUT "EINEN BUCHSTABEN
   EINGEBEN";B$
20 W=ASC(B$)
30 PRINT "DER BUCHSTABEN "B$"
   BESITZT DEN ASCII-WERT "B
40 GOTO 10
```

Wir starten unser Beispiel:
RUN <RETURN>
A <RETURN>
65
B <RETURN>
66
7 <RETURN>
55

Wenn wir in der ASCII-Tabelle (Teil 1) nachschauen, so werden wir erkennen, was die ausgegebenen Zahlen zu bedeuten haben. Der ASCII-Wert vom Buchstaben „A" beträgt somit 65, der von „B" beträgt 66 usw.
Interessant ist es auch, wenn wir einmal statt einem Buchstaben ein komplettes Wort eingeben:

RUN <RETURN>
OTTI
79

Wie aus dem Beispiel zu erkennen ist, wird in diesem Fall nur der erste Buchstabe der Zeichenkette (String) beachtet. Der Wert 79 entspricht dem Buchstaben „0".



Mensch Vati, Du immer mit den „blöden" Hausaufgaben, der Basic-Kurs in Computronic ist doch viel wichtiger!

CHR$(...)
Die entgegengesetzte Funktion zu ASC ist die CHR$-Funktion. Diese ermittelt aus einer Zahl das alphanumerische Zeichen. So können wir aus dem Wert 79 wieder eine „0" werden lassen:

PRINT CHR$(79) <RETURN>
0

Leider besitzen im ASCII-Code nur die Zahlenwerte von 32 bis 127 allgemeine Gültigkeit zur Darstellung von alphanumerischen Zeichen. Die Zahlenwerte von 0 bis 31 stellen in der Regel Steuerzeichen dar. Da diese jedoch nicht genormt sind, verwenden die einzelnen Computer unterschiedliche Vereinbährungen. Auch die Zahlenwerte von 128 bis 255 sind nicht genormt, werden jedoch häufig für Grafik- oder reverse Zeichen benutzt. Mit dem folgenden Programm können wir uns eine komplette ASCII-Code Tabelle auf dem Schirm ausgeben lassen. Die Werte unter 32 und über 127 werden aus den erwähnten Gründen nicht beachtet. Trotz dieser Einschränkung ist das Beispiel nicht auf allen Rechnern läuffähig, da einige Computer von der ASCII-Norm abweichen und ebenfalls Steuerzeichen in diesen Bereich unterbringen. Ein Beispiel sind die Commodore-Rechner, die nach der Ausgabe eines Anführungszeichens oft nicht mehr fehlerfrei arbeiten.

```
10 REM BEISPIEL 7.1
100 PRINT"ASCII-TABELLE COMPUTRONIC-KURS"
110 PRINT"------------------------------"
120 PRINT
130 FOR Z= 0 TO 15
140 FOR S= 2 TO 7
150 IF S*16+Z>95 THEN 180
160 PRINT TAB(S*6-11);S*16+Z;TAB(S*6-7);CHR$(S*16+Z);
170 GOTO 190
180 PRINT TAB(S*7-17);S*16+Z;TAB(S*7-12);CHR$(S+16+Z)
190 NEXT S
200 PRINT
210 NEXT Z
READY.
```

Das erste sinnvolle Programm:

Umrechnung römischer Zahlen

Wir haben inzwischen die wichtigsten Eigenschaften und Möglichkeiten eines Basic-Interpreters kennengelernt. Wir wollen nun unsere Kenntnisse dazu verwenden, ein kleines Umwandlungsprogramm zu schreiben.
Das römische Zahlensystem ist nicht wie unser Dezimalsystem ein Stellenwertsystem mit einer bestimmten Wertigkeit je Stelle, sondern die einzelnen Ziffern haben einen Wert:

I = 1
V = 5
X = 10
L = 50
C = 100
D = 500
M = 1000

Bei der Umrechnung in das Dezimalsystem müssen wir deshalb werten, ob bei der römischen Zahl benachbarte Ziffern größer oder aber kleiner sind. Denn nur so kann man entscheiden, ob der Wert der Ziffer addiert oder subtrahiert wird.
Mit der ersten FOR...NEXT Schleife wird die römische Zahl „zerlegt", und die in ihr vorkommenden Wertigkeiten werden dem Feld A zugeordnet. In der zweiten Schleife wird dann die Dezimalzahl ermittelt, indem man aufgrund der Anfrage in Zeile 280 entscheidet, ob addiert oder aber subtrahiert wird.
Um Ihr Verständnis zu prüfen, sollten Sie sich einmal die Funktion an der römischen Zahl MCMLXXXIV klarmachen.

```
SEARCHING FOR BASIC72

100 REM ROEMISCHE ZAHLEN
110 REM BEISPIEL 7.2
120 REM
130 INPUT "ROEMISCHE ZAHL ";R$
140 L=LEN(R$)
150 FOR I=1 TO L
160 Z$=MID$(R$,I,1)
170 IF Z$="I" THEN A=1
180 IF Z$="V" THEN A=5
190 IF Z$="X" THEN A=10
200 IF Z$="L" THEN A=50
210 IF Z$="C" THEN A=100
220 IF Z$="D" THEN A=500
230 IF Z$="M" THEN A=1000
240 A(I-1)=A
250 NEXT I
260 REM DEZIMALZAHL BERECHNEN
270 FOR I=L-1 TO 0 STEP-1
280 IF A(I)<A(I+1) THEN D=D-A(I):GOTO 290
285 D=D+A(I)
290 NEXT I
300 REM ERGEBNISAUSGABE
310 PRINT R$;" =";D
320 END
READY.
```

Rechnen mit Buchstaben und Zeichen

Neben den bisher aufgezeigten Stringfunktionen gibt es noch einige weitere interessante Möglichkeiten. Da es öfters vorkommt, daß in einer Stringvariable eine Zahl enthalten ist, existiert eine Funktion, die dieser Zahl einem String entnimmt und einer Zahlen-Variable zuweist. Dies ist die VAL-Funktion.

PRINT VAL("100") <RETURN>
100

Auf den ersten Blick scheint es, als habe die VAL-Funktion keine Bedeutung, denn die Zahl 100 wäre auch ohne diese Funktion ausgegeben worden. Das nächste Beispiel zeigt schon etwas deutlicher die Funktion von VAL:

PRINT VAL("100DM") <RETURN>
100

Wir wollen auch hier ein kleines Beispielprogramm schreiben:

```
10 REM Beispiel 7.3
20 INPUT "STRING ";S$
30 Z=VAL(S$)
40 PRINT "DIE ZAHL LAUTET "Z
50 GOTO 20
```

Wir starten ...

```
RUN <RETURN>
STRING ? ICH HABE 100 DM
DIE ZAHL LAUTET 0
STRING ? 100 DM HABE ICH
DIE ZAHL LAUTET 100
STRING ? 123.45 DM SCHULDEN
123.45
```

Wie zu erkennen ist, löst VAL nur die Zahlen heraus, welche direkt am Anfang des String liegen. Weitere eventuell auftretende Zahlen werden im String nicht berücksichtigt. Das ganze können wir auch umdrehen. In Basic existiert ebenfalls eine Funktion zum Umwandeln einer Zahl oder Zahlenvariable in einen String. Diese Funktion heißt STR$ und wird wie folgt angewendet:

PRINT STR$(1986)
1986

Auch bei diesem Beispiel ist auf den ersten Blick keine Funktion zu erkennen, da eine Zahl sich im Aussehen nicht von einem String unterscheidet. Die nachfolgende Programmzeile zeigt die Funktion etwas deutlicher:

PRINT STR$(1986)+"TEST"
1986TEST

Es versteht sich von selbst, daß eine mit STR$ umgewandelte Zahl danach mit al-

len anderen Stringfunktionen wie LEFT$, RIGHT usw. bearbeitet werden kann. Sie werden sich jetzt sicher fragen, wozu die Umwandlungen eigentlich gut sind. Ein Anwendungsfall wäre beispielsweise die Entwicklung einer benutzerfreundlichen und abbruchsicheren Eingaberoutine. Wir werden uns damit in einer der nächsten Folgen unseres Basic-Kurses noch genauer beschäftigen.

Literaturquellen:

Grundkurs in Basic, Ulrich Strobel, Sybex Verlag
Basic Brevier, Siegmar Wittig, Heinz Heise Verlag
Commodore Handbücher
Spectrum Handbuch
Commodore 116/16/plus4, Ekkehard Kaier, Vieweg Verlag

Basic Kurs Teil 8 folgt in der nächsten Computronic!

BASIC-KURS - Teil 7 ------- ● ------- BASIC-KURS - Teil 7

Der COMMODORE-SONDERTEIL

FIGHT NIGHT

Ein attraktives Sportprogramm für Ihren Commodore 64

Fight Night ist ein Boxspiel für 2 Personen. Der spannende Kampf geht über 10 Runden. In den Rundenpausen werden die Punkte angezeigt. Und zwar die Punkte der letzten Runde, die Gesamtpunktezahl, und die Kraft, die der Boxer noch zur Verfügung hat. Ist diese kleiner als Null, geht der Boxer K.O.
Zuletzt wird noch die Ausdauer angegeben, die sich mit jedem Schlag verringert. Deshalb ist es nicht sinnvoll, wild auf seinen Gegner einzuschlagen. Denn läßt die Ausdauer nach, muß sich der Boxer zwischen den Schlägen länger erholen.
Sollten nach 10 Runden beide Boxer noch kampffähig sein, wird der Sieger nach Punkten bestimmt.

Steuerung:

Die Figuren werden mit den beiden Joysticks gesteuert. Dabei sind folgende Bewegungen möglich:
links-rechts gehen (Joystick in die entsprechende Richtung)
oben/unten decken (Joystick oben und unten)
oben/unten schlagen (Joystick oben/unten + Feuerknopf)
Als Treffer zählt nur der Schlag, der die ungedeckte Seite des Gegners trifft.

Action

FIGHT NIGHT ist ein echtes Topprogramm! Hier wird nicht wildes „Drauflosdreschen" gefordert, sondern taktisch geschicktes Kampfverhalten und kluges Einteilen der Kräfte verlangt.

Spannung/Dramatik/aufregendes Kampfgeschehen

Ring frei! Sie sind der Champion und müssen Ihre Ausdauer und Geschicklichkeit einsetzen, um Ihren Titel zu verteidigen.

BASIC-KURS Teil 8

Im letzten Teil unseres Kurses haben wir zahlreiche Stringfunktionen kennen gelernt. Wir wollen nun unsere Kenntnisse der Stringverarbeitung vervollständigen und durch einige Beispielprogramme den Einstieg in die Kunst des Programmierens erlernen. Gerade für das Programmieren von Spielprogrammen sind gute Kenntnisse in der Stringverarbeitung unentbehrlich.

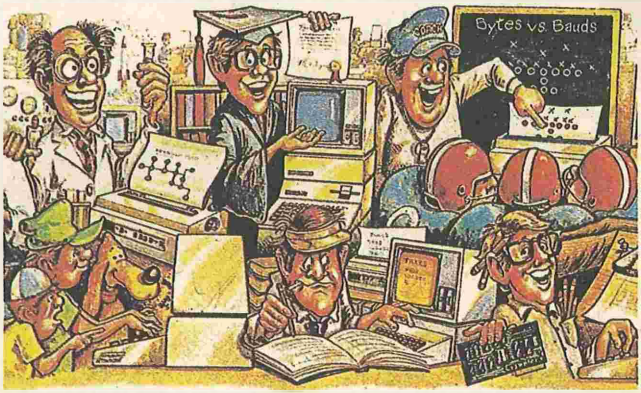
Wir wollen uns als erstes die Aufgabe stellen, ein Programm zu entwickeln, in dem ein Objekt durch zwei Tasten horizontal bewegt wird. Um diese Aufgabe zu bewältigen, benötigen wir folgende Befehle:

1. Ausgabe -Befehl für den Bildschirm
2. Tastaturabfrage
3. Möglichkeit vertikal zu positionieren
4. Möglichkeit horizontal zu positionieren

Punkt 1 ist die leichteste Aufgabe. Hier nehmen wir den schon besprochenen PRINT-Befehl.Beim zweiten Punkt wird es schon schwieriger. Bisher haben wir neben INPUT noch keine weitere Möglichkeit der Tastaturabfrage kennengelernt. Da INPUT im Prinzip zwar funktionieren würde, jedoch immer ein abschließendes RETURN benötigt, verwenden wir hier einen neuen Befehl: **GET**! In vielen Basic-Versionen wird statt des GET-Befehles auch der INKEY$ - Befehl verwendet. Wir werden deshalb hier beide Befehle genauer untersuchen.

GET Die Tastaturabfrage

Wenn wir noch einmal zurückdenken und die INPUT-Anweisung betrachten, so wissen wir, das mit dieser numerische Werte oder Zeichenketten eingegeben und einer Variablen zugewiesen werden. Solange die Eingabe, die auch aus mehreren Zeichen bestehen darf, nicht abgeschlossen ist, muß man auf eine Taste drücken (RETURN,CR,ENTER o.ä., je nach Computertyp). Der GETY-Befehl arbeitet hier etwas anders, er erlaubt nur die Eingabe eines einzelnen Zeichens, wodurch auch das Betätigen einer RETURN-Taste entfällt. Ein weiterer Vorteil ist, das GET nicht unbedingt auf eine Taste wartet, das bedeutet auch, wenn keine Taste betätigt wurde, arbeitet das Programm weiter. Achtung, einige Basic-Interpreter machen auch hier eine Ausnahme, was allerdings der Normung widerspricht. Der Vorteil, das der Programmablauf nicht angehalten wird,



kann jedoch schnell zum Nachteil werden, wenn eine Taste nicht im richtigen Augenblick betätigt wird. Bei vielen Computern hilft hier auch nicht das Festhalten einer Taste, da diese durch ein Prellschutzverfahren abgeschaltet wird.

Ein kleines Beispiel zeigt dieses Problem:

```
10 PRINT "GEBE ZIFFER EIN"
20 GETZ
30 PRINT "DIE ZIFFER LAUTET:";Z
40END
```

Der Ablauf ist relativ einfach zu verstehen. Nach dem Start werden Sie aufgefordert, eine Ziffer einzugeben. In Zeile 20 wird nun die eingegebene Ziffer einer Variablen zugeordnet. In Zeile 30 wird schließlich diese Zahl auf dem Schirm ausgegeben. Das Problem besteht nun darin, das in Zeile 20 nicht auf eine Betätigung einer Taste gewartet wird. Wurde Ihre Taste also nicht rechtzeitig betätigt, wird immer eine 0 ausgegeben. Probieren Sie es einfach einmal. Sie werden sehen, das es fast unmöglich ist, nach dem Programmstart noch eine Ziffer einzugeben, bevor das Programm beendet ist.

Wie hilft man sich hier ? Ganz einfach, wir führen eine IF-THEN Bedingung durch, welche überprüft, ob etwas eingegeben wurde. Ist dies nicht der Fall, so wird die Tastaturabfrage einfach wiederholt. Erst wenn wirklich ein Zeichen eingelesen wurde, wird das Programm fortgesetzt. In der Praxis würde unser Programm wie folgt aussehen:

```
10 PRINT "BITTE ZIFFER EINGEBEN"
20 GETZ
30 IF Z=0 THEN 20
40 PRINT "DIE ZIFFER LAUTET:"Z
50END
```

Die Funktionsweise ist relativ leicht zu verstehen. In Zeile 30 wird abgefragt, ob eine Ziffer eingegeben wurde. Ist dies nicht der Fall, oder war die Ziffer 0, so wird die Abfrage wiederholt. Nachteil dieser Methode ist, das niemals eine 0 eingegeben werden darf. Auch die Eingabe eines Buchstäben wird zum Abbruch oder zur Fehlermeldung führen. Um diese Nachteile zu beheben, kann man den GETY-Befehl auch für Stringvariablen verwenden.

Beispiel:

```
10 PRINT "BITTE GEBE BUCHSTABEN
EIN"
20 GET B$
30 IF B$="" THEN 20
40 PRINT "DER BUCHSTABE WAR:"B$
50END
```

Die Arbeitsweise entspricht vollkommen unseren vorangegangenen Beispielen, mit der Außnahme, das hier jedes beliebige Zeichen eingegeben werden darf. Ziffern werden in diesem Fall als Zeichen und nicht als Zahl betrachtet. Da wir jedoch eine Zahl eingeben wollten, müssen wir anders vorgehen:

```
10 PRINT "BITTE GEBE ZIFFER EIN"
20 GET B$
```

```
30 IF B$=""  THEN 20 :REM WURDE
NICHTS EINGEGEBEN ?
40 Z=VAL(B$):REM STRING WIRD IN
ZAHL GEWANDELT
50 PRINT "DIE EINGEGEBENE ZAHL
LAUTET:"Z
60 END
```

Die Zeile 40 wandelt einfach eine Stringvariable in eine Zahl um, wodurch alle unsere Probleme gelöst werden. Durch die Zeile 30 wird erreicht, das nur dann die Eingabe wiederholt wird, wenn wirklich kein Zeichen eingegeben wurde. Nun wollen wir die Eingabe nach zwei verschiedenen Bedingungen überprüfen:

```
60000 PRINT "WOLLEN SIE NOCHMAL
SPIELEN ?"
60010 PRINT "GEBEN SIE J FÜR JA
UND N FÜR NEIN EIN"
60020 GET B$
60030 IF B$=""  THEN 60020
60040 IF B$="J" THEN RUN
60050 IF B$="N" THEN PRINT "BIS
BALD !": END
60060 GOTO 60020:REM EINGABE
WIEDERHOLEN
```

Das obige Beispiel könnte am Ende eines Basic-Programmes stehen. Es fragt sie, ob sie noch einmal spielen möchten. Antworten Sie mit J, so wird das Programm neu gestartet. Antworten Sie mit N, so verabschiedet sich das Programm. Wurde keine der beiden Tasten betätigt, wird die Abfrage wiederholt.

INKEY Eine ähnliche Tastaturabfrage

Bei vielen Rechnern, wie beispielsweise beim Schneider CPC existiert keine GET Anweisung. Hier muß diese durch die Funktion INKEY ersetzt werden. Die Funktion ist fast identisch mit GET, jedoch ist die Syntax etwas anders. Das nachfolgende Beispiel demonstriert dies sehr deutlich.

```
10 PRINT "WOLLEN SIE NOCHMAL
SPIELEN (J/N) ?"
20 B$=INKEY$
30 IF B$=""  THEN 20
40 ...:REM Fortsetzung wie oben
```

Nun aber genug über die Tastaturabfrage vom Basic aus. Das eine Problem ist damit gelöst. Wir gehen nun zum Punkt 3 über: Vertikales Positionieren. Dieses Problem wird auf den Computern sehr unterschiedlich gelöst. Teilweise gibt es Befehle, welche den Cursor direkt positionieren können, wie LOCATE,PRINT AT oder ähnliches, andererseits existieren bei einigen Rechnern Steuerzeichen für die Cursorbewegung. Die LOCATE-Anweisung, wie sie beim CPC 464 verwendet wird, positioniert den Cursor an eine beliebige Bildschirmstelle:

```
LOCATE 5,10: PRINT "HALLO"
```

Diese Befehlsfolge würde das Wort "HALLO" in die fünfte Spalte der 10ten Zeile setzen. Durch das Verwenden von Variablen läßt sich der Locate-Befehl sowohl zum vertikalen, als auch zum horizontalen Positionieren des Cursors verwenden. Damit wäre beispielsweise Punkt 3 und 4 beim Schneider CPC 464 erfüllt. Auch bei vielen anderen Rechnern wird ein ähnlicher Befehl genutzt, meist unterscheidet sich dieser nur vom Namen, jedoch nicht in der Sntax. Bei den Commodore Rechnern ist dieses Problem schon etwas größer. Wir haben schon in einem der letzten Teile unseres Kurses eine Routine vorgestellt, welche uns den Cursor mit Hilfe von Steuerzeichen positioniert. Wir wollen jetzt noch einmal etwas genauer auf diese sogenannten Steuerzeichen eingehen.

Steuerzeichen können beim Commodore nur innerhalb von Anführungszeichen eingegeben werden. Das bedeutet, nach der Eingabe eines Anführungszeichens befindet sich der Rechner in einem anderen Modus. Nun können weder Farben gewechselt, noch Cursorbewegungen durchgeführt werden. Beispielsweise würde das Betätigen der Cursortasten unterschiedliche Steuerzeichen auf den Schirm bringen. Diese Steuerzeichen veranlassen später im Programm den Rechner den Cursor zu versetzen, oder die Farbe zu wechseln.

In dem obigen Beispielprogramm wird der Cursor mit der Hilfe der Steuerzeichen, in die 5te Spalte der Zeile 10 gesetzt. Dort wird dann in Zeile 70 das Wort "HALLO" ausgegeben. Wie zu erkennen ist, gibt es auch noch andere Steuerzeichen, in Zeile 40 zum Beispiel zum Bildschirmlöschen. Für die Besitzer eines Commodore-Rechners gibt eine Tabelle Auskunft, die wir in der nächsten Computronic-Ausgabe (BASIC-KURS Teil 9) veröffentlichen werden.

Mensch Vati, Du immer mit den „blöden" Hausaufgaben, der Basic-Kurs in Computronic ist doch viel wichtiger!

Literaturquellen:

Grundkurs in Basic, Ulrich Stroebel, Sybex-Verlag

Basic Brevier, Siegmar Wittig. Heinz Heise Verlag

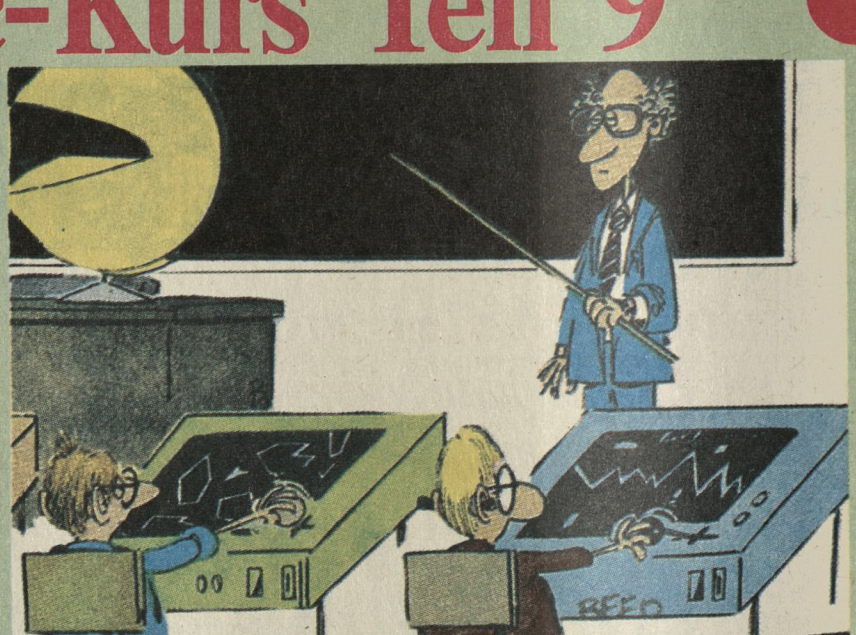
Commodore Handbuch
Spectrum Handbuch
Commodore 116/16/plus 4, Ekkehard Kaiser, Vieweg Verlag

Autor des BASIC-KURSES
Frank Brall

* * *

```
10 REM BEISPIEL 8.1
20 REM CURSOR MIT STEUERZEICHEN
30 REM POSITIONIEREN
40 PRINT"▒";:REM BILDSCHIRM LOESCHEN
50 PRINT"▒▒▒▒▒▒▒▒▒▒";:REM CURSOR RUNTER
60 PRINT"▒▒▒▒▒";:REM CURSOR RECHTS
70 PRINT"HALLO"
80 END
```

Basic-Kurs Teil 9



Im letzten Teil unseres Kurses haben wir gelernt, wie wir mit Steuerzeichen den Cursor versetzen können.
Steuerzeichen lassen sich zwar einfach programmieren, haben jedoch den Nachteil, daß sie den Überblick nicht gerade fördern.
Außerdem ist es sehr schwierig, Variablen einzusetzen, um den Cursor zu positionieren. Eine Tabelle aller Commodore-Steuerzeichen ist in diesem Heft unter der Rubrik Checksummer abgedruckt.
In unserem Projekt werden wir die Steuerzeichen nur zum vertikalen Positionieren einsetzen. Für die horizontale Positionierung verwenden wir einen neuen Befehl: TAB

TAB Horizontales Positionieren

Durch den TAB-Befehl hat man die Möglichkeit, die Ausgabe eines Zeichens oder einer Zahl an einer beliebigen Spalte vorzunehmen. Die TAB-Funktion, ist kein alleine arbeitender Befehl, sondern eine Funktion, die immer nur im Zusammenhang mit einer PRINT-Anweisung funktioniert. Das folgende Beispiel demonstriert die Syntax dieses Befehles:

```
10 INPUT „SPALTE";X
20 PRINT TAB(X)„TEXT"
30 END
```

Die TAB-Funktion steht hinter der PRINT-Anweisung. Die nachfolgende Zahl gibt an, in welche Spalte der nachfolgende Text geschrieben werden soll. Der TAB-Wert stellt also den Bezug zwischen Zeilenanfang und der tatsächlichen Textzeile her. Ist die tatsächliche Spalte höher als das Parameter der TAB-Anweisung, so wird diese ignoriert und der Text in die nächst mögliche Spalte geschrieben. Auch dies läßt sich durch ein Beispiel demonstrieren:

```
10 PRINT „DAS IST EIN TABULATOR
          BEISPIEL";
20 PRINT TAB(10) „TEST"
```

In Zeile 10 dieses Beispiels, wird ein Satz auf dem Bildschirm ausgegeben. Das Semikolon am Ende der Zeile bewirkt, daß keine Zeilenschaltung erfolgt und der Cursor praktisch am Zeilenende stehen bleibt. Die nachfolgende TAB-Funktion versucht nun, das Wort „TEST" in Spalte 10 zu setzen. Da der Cursor jedoch weit über diese Spaltenposition hinaus bewegt wurde, wird diese Anweisung ignoriert und das Wort „TEXT" an den bereits ausgegebenen Satz angesetzt.

Das nachfolgende DEMO-Programm veranschaulicht noch einmal die Möglichkeiten der TAB-Funktion. Dieses DEMO ist beispielsweise als Vorspann bzw. Titel-Effekt zu einem Spiel denkbar.

```
10 REM DEMO FUER TAB-EFFEKTE
20 FOR X= 1 to 28
30 PRINT TAB(X)„DEMO TEIL 8"
40 NEXT X
50 FOR X=28 to 1 STEP-1
60 PRINT TAB(X)„DEMO"
70 NEXT
80 GOTO 20
```

POS Position des Cursors

Wie auch der TAB-Befehl, wirkt die POS-Funktion nur im Zusammenhang mit der Bildschirmausgabe. Sie zeigt die Position des Cursors innerhalb der aktuellen Schreibzeile an. Das ist immer die Zeile, in der die letzte Ausgabe erfolgte. Wenn wir also relativ zu dieser letzten Ausgabe, eine weitere Ausgabe innerhalb der gleichen Bildschirmzeile durchführen wollen, brauchen wir nur den Wert, der von POS geliefert wird, zum Abstand für die nächste Ausgabe hinzu addieren. Der POS-Befehl gehört zum Standard eines Basic-Dialektes, unterscheidet sich jedoch teilweise durch die Syntax. In der Regel ist POS eine „DUMMI-Funktion". Dies bedeutet, daß der zu übergebende Parameter bei dieser Funktion keine Bedeutung hat, sondern lediglich aus formalen Gründen vorhanden sein muß.
Die Syntax sieht also folgendermaßen aus:
PRINT POS(0)
Selbstverständlich kann POS auch als Teil einer Berechnung benutzt werden, und muß somit nicht unbedingt hinter einer Ausgabeanweisung wie PRINT verwendet werden. In dem nachfolgenden Beispiel gibt die POS-Funktion die X-Position des „*"-Zeichen's an.

```
10 FOR X=1 to 10
20 PRINT TAB(X)„POSITION";
30 PRINT „*";POS(0)
40 NEXT X
```

SPC Horizontales Positionieren etwas anders

Die SPC-Funktion arbeitet auf ähnliche Weise wie die TAB-Funktion. Jedoch wird

hier die Position nicht relativ zum Zeilenanfang berechnet, sondern relativ zur letzten Cursor-Position. Dabei werden dazwischenliegenden Positionen übersprungen, aber nicht durch Leerzeichen überschrieben. Ersetzen wir in unserem obigen Beispiel die TAB-Funktion durch die SPC-Funktion, so werden wir keine Veränderung beim Ablauf des Programmes bemerken. Dies ergibt sich dadurch, daß vor Aufruf der SPC-Funktion der Cursor am Zeilenanfang steht. Die SPC-Funktion unterscheidet sich also nur dann von der TAB-Funktion, wenn sich der Cursor nicht am Zeilenanfang befindet. In unserem nächsten Beispiel erreichen wir dies, indem wir hinter der PRINT-Anweisung ein Semikolon setzen. Die PRINT-Anweisung wird dadurch angewiesen keinen Zeilenvorschub durchzuführen, wodurch der Cursor praktisch auf der letzten Schreibstelle stehen bleibt.

```
10 FOR X= 1 TO 10
20 PRINT SPC(X)„A";
30 NEXT X
```

Diese 3 Zeilen haben zur Folge das der Buchstabe „A" in steigenden Abständen voneinander ausgegeben wird:

```
RUN return
A..A...A....A.....A......A.......A.......
A.........A.........A
```

Zurück zu unserem Projekt

Errinnern wir uns noch einmal an unser Vorhaben. Wir wollten ein kleines Programm schreiben, welches durch Tastenbetätigung ein Objekt horizontal bewegt. Wir haben nun eine Reihe wichtiger Befehle kennen gelernt, und können damit alle Probleme, die dieses Projekt aufwirft, auf einfachste Weise lösen. Als Objekt nehmen wir in unserem Programm den Bindestrich „-". Die Vertikale Positionierung nehmen wir einmal mit Steuerzeichen und ein zweites Mal mit der LOCATE-Anweisung vor. Zur horizontalen Verschiebung verwenden wir in den ersten und dritten Beispiel die TAB-Funktion, und im zweiten Beispiel ebenfalls die LOCATE-Anweisung. Das bedeutet in der Praxis, das die ersten beiden Beispiele auf dem CPC, und das dritte Beispiel auf dem C64 lauffähig sind. Unser Objekt („-") läßt sich mit den Tasten „+" und „-" beliebig in der Zeile bewegen. Wir haben mit diesen Beispielen den Grundstein eines Spieles kennen gelernt.Versuchen Sie einmal, durch Ändern einiger Parameter den Ablauf des Programmes zu beeinflussen.

Einige Programmierbeispiele:

```
10 REM *** beispiel 1 ***
20 CLS
30 x=20 :REM anfangsstellung
40 e$=INKEY$ :REM tastaturabfrage
50 IF e$="+" THEN x=x+1
60 IF e$="-" THEN x=x-1
70 IF x>37 THEN x=37
80 IF x<0  THEN x=0
90 LOCATE 1,19 :REM vertikal position
100 PRINT TAB(x)" - "
110 GOTO 40
```

```
10 REM *** beispiel 2 ***
20 CLS
30 x=20 :REM anfangsstellung
40 e$=INKEY$ :REM tastaturabfrage
50 IF e$="+" THEN x=x+1
60 IF e$="-" THEN x=x-1
70 IF x>37 THEN x=37
80 IF x<1  THEN x=1
90 LOCATE x,19 :REM x und y position
100 PRINT " - ";
110 GOTO 40
```

```
10 REM *** BEISPIEL 3 ***
20 PRINT "␣"; :REM BILDSCHIRM LOESCHEN
30 X=20 :REM ANFANGSSTELLUNG
40 GET E$ :REM TASTATURABFRAGE
50 IF E$="+" THEN X=X+1
60 IF E$="-" THEN X=X-1
70 IF X>37 THEN X=37
80 IF X<0 THEN X=0
90 PRINT"␣␣␣␣␣␣␣␣␣␣␣␣";:REM VERTIKAL POSITION
100 PRINT TAB(X)" - ";
110 GOTO 40
READY.
```

Schlußwort

An dieser Stelle möchten ich meinen Basic-Kurs beenden. Wir haben in 9 Teilen unseres Kurses versucht, auch dem absoluten Laien den Einstieg in die Programmiersprache Basic zu erleichtern. Zwar konnten wir in diesem Kurs nicht auf alle Möglichkeiten dieser Sprache eingehen, konnten Ihnen aber sicherlich einen kleinen Überblick darüber geben, was in dieser Sprache steckt. Wer Spaß an diesem Kurs gefunden hat, und sich weiterhin mit Basic befassen möchte, dem empfehle ich die unten aufgeführte Literatur. Übrigens, zum Üben eignen sich auch Listings, die regelmäßig in diesem Heft abgedruckt werden. Vieleicht versuchen Sie einmal ein abgedrucktes Spiel zu verbessern, oder auf einen anderen Rechner umzuschreiben. Ich wünsche Ihnen viel Erfolg!

Frank Brall

Literatur:

Grundkurs in Basic, Ulrich Stroebel, Sybex Verlag
Basic Brevier, Siegmar Wittig, Heinz Heise Verlag
Commodore 116/plus 4, Ekkehard Kaiser, Vieweg Verlag
Schneider CPC Basic Wegweiser, Ekkehard Kaier, Vieweg Verlag
Das große Commodore Basic Handbuch, Michael Orkim, Sybex-Verlag
Basic, W. Meyer / K.Schacht, Hanser Verlag
Das Standard Basic-Buch zum CPC, E. Unger, Heim-Verlag
Wie arbeite ich mit dem CPC 464, Wolfgang Schneider, Vieweg Verlag

Einführung/Neuerungen/und Programme in TURBO-PASCAL

Endlich gibt es eine Pascal-Version, die auf (fast) jedem Computer lauffähig ist. Der Welterfolg von Turbo-Pascal hat tausenden von Computerfreunden die Chance eröffnet, leistungsstarke und professionelle Programme zu entwickeln, ohne die Nachteile älterer Pascalcompiler in Kauf nehmen zu müssen. Noch nie hat in der Geschichte der EDV ein Compiler eine derartige Verbreitung und Reaktion unter den Anwendern hervorgerufen. Wir haben uns deshalb entschlossen die Pascal-Freunde nicht im Stich zu lassen, sondern mit Information, Tips und Programmen zu versorgen. Wer noch nicht so vertraut mit dem neuen Pascal-Dialekt ist, hat Gelegenheit in einem mehrteiligen Kurs sein Wissen zu erweitern. Autoren von Turbo-Pascal Programmen sind hiermit aufgerufen, uns mit Programmen und Tips zu unterstützen (natürlich gegen Honorar).

Rüdiger Pfister und Roland Schweizer, die Autoren unseres neuen Turbo-Pascal Kurses, haben ihr Konzept derart aufgebaut, daß vor allem Anfänger und leicht Fortgeschrittene angesprochen werden. Außerdem soll eine Integration von allgemeinem Programmierkurs und Einführung in ein spezielles Programmsystem (nämlich Turbo-Pascal) erreicht werden. Und nun viel Spaß mit Turbo-Pascal Teil 1.!

Einführung Turbo-Pascal

Pascal wurde 1970 von N. Wirth in Zürich entwickelt. Die Programmiersprache hat sich sehr schnell als Ausbildungssprache (in Universität und Schule) wie auch als Sprache zur Bewältigung technisch-wissenschaftlicher Probleme etabliert.
Es gibt inzwischen verschiedene Dialekte von Pascal. Turbo-Pascal existiert seit 1983 und die Version 3.0 von Turbo-Pascal seit 1985.
Welche Gründe sprechen eigentlich dafür, sich mit Pascal, speziell mit Turbo-Pascal, zu beschäftigen?
Für Pascal wird oft als Argumente angeführt:

- es bietet die Möglichkeit, Strukturen von Problemen klar herauszuarbeiten und zu verdeutlichen
- es erlaubt gut lesbare und verständliche Programme
- es ist allgemein anwendbar (im Gegensatz zu Programmiersprachen, die für spezielle Anwendungen entwickelt wurden)
- es erzieht zu einem guten Programmierstil
- die Datenstruktur kann an die zu lösenden Probleme angepaßt werden
- es ist flexibel und effizient
- es ist leicht erlernbar (dies gilt bestimmt nicht im Vergleich zu BASIC, aber im Vergleich zu anderen Programmiersprachen)
- Man kann Programme aus sehr kleinen und deshalb sehr übersichtlichen Teilen (sogenannten Modulen) zusammensetzen

Diese Punkte sollen zunächst so aufgezählt stehen bleiben. Was diese im einzelnen bedeuten, wird im Verlauf des Kurses an konkreten Beispielen deutlich werden.
Für Turbo-Pascal als Dialekt spricht, daß es relativ preiswert ist, sehr einfach zu bedienen ist (obwohl es ja ein Compiler ist) und im großen und ganzen dem Standard-Pascal entspricht. Turbo-Pascal gibt es für verschiedene Rechner: CP/M, MS-DOS, PC-DOS.
Bevor die praktische Arbeit beginnt, noch ein paar kurze Bemerkungen zu dem, was wir jetzt ständig erstellen wollen:

Was ist ein Programm?

Um verstehen zu können, wozu das Schreiben von Programmen sinnvoll ist, wollen wir zunächst einige allgemeine Überlegungen voranstellen, aus denen klar werden soll, welchen Zweck Programme erfüllen können. Ein für den Anfang sehr nützlicher Vergleich ist der mit einer Maschine oder einem Werkzeug, wie jede Maschine bearbeitet auch ein Programm ein bestimmtes Material: dieses Material sind die Daten. Unter Daten kann man sich alle möglichen Symbole oder Zeichen vorstellen - Buchstaben, Sätze, Zahlen usw. -, wir wollen uns zunächst der Einfachheit halber auf die Bearbeitung von Zahlen beschränken.
Die Maschine selbst, eben das jeweilige Programm, ist aus bestimmten elementaren Bauteilen konstruiert: Anweisungen, Zuweisungen usw., allgemein gesagt aus einer Menge von Befehlen. Je nachdem welchen Zweck die Maschine (das Programm) erfüllen soll. Solch eine Folge von Befehlen wird auch Algorithmus genannt.
Bildlich kann man sich den Verarbeitungsvorgang so vorstellen, daß das Rohmaterial, die Ausgangsdaten, oben in die Programmaschine hineingeschüttet (Eingabe), darin bearbeitet (Verarbeitung) und schließlich unten die veränderten Daten ausgeworfen werden (Ausgabe). Diese grundsätzliche Reihenfolge wird auch EVA-Prinzip (Eingabe -> Verarbeitung -> Ausgabe) genannt.
Den Vorgang der Konstruktion einer solchen Datenverarbeitungsmaschine auf einem Computer nennt man Programmierung, die Mittel, die man dazu neben dem Computer selbst (der Hardware) benötigt, sind die verschiedenen Programmiersprachen (Software) wie z. B. BASIC, PASCAL oder LOGO.
Soweit einige allgemeine einführende Überlegungen. An einem einfachen Beispiel soll nun verdeutlicht werden, wie die Arbeitsweise eines Programms ko

TURBO PASCAL – Der Kurs *(Teil 1)*

kret aussehen kann.
Bevor es losgeht noch eine Frage: Habt Ihr von der Turbo-Pascal-Diskette eine Sicherungskopie erstellt? Wenn nicht, dann holt das bitte vorher noch nach. Manchmal passiert es doch, daß die Daten auf der Diskette beschädigt werden und wenn das bei der Originaldiskette passiert ist der Ärger und die Lauferei groß.
Bevor das erste Programm eingegeben werden kann, muß zuerst mal „Turbo-Pascal" in den Rechner geladen werden. Ich gehe davon aus, daß der Rechner eingeschaltet und das entsprechende Betriebssystem (MS-DOS, CPM etc.) geladen ist. Legt die Kopie der Turbo-Pascal-Diskette ins Laufwerk (bei zwei Laufwerken ins Laufwerk A) ein, gebt „Turbo" ein mit abschließendem Drücken der <Enter>-Taste. Nach kurzer Zeit müßte euer Schirm folgendermaßen aussehen:

Schirm-Meldung 1

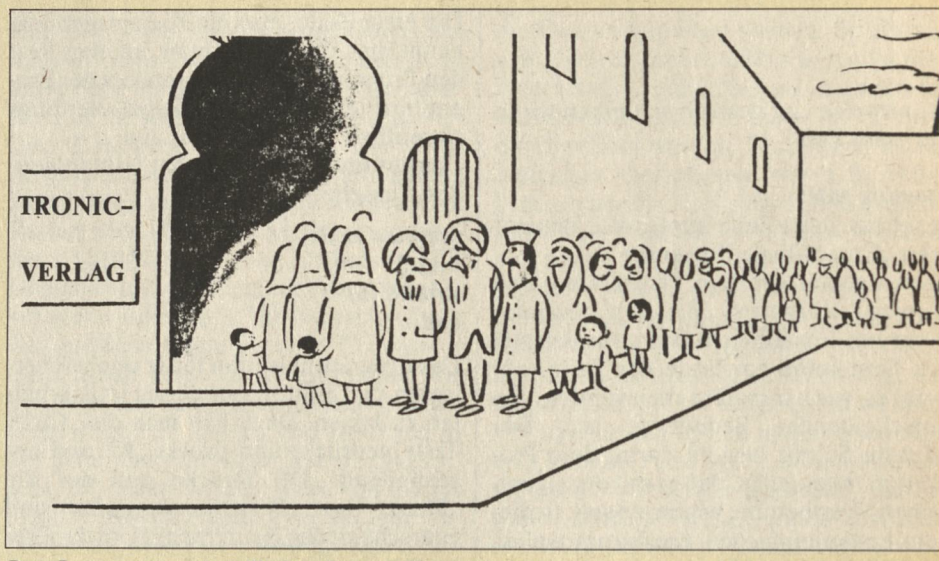
```
TURBO Pascal system Version 3.01A
                              PC-DOS
Copyright ©
1983,84,85                BORLAND Inc.

b/w display 80 x 25

Include error messages (Y/N)?
```

Auf die Meldung „Include error messages (Y/N)" ein „Y" (ohne <Enter>) eingeben. Das bedeutet, daß Fehlermeldungen nicht nur als Nummern angegeben werden, was für den Anfang ganz nützlich ist. Danach wird automatisch „Turbo.MSG" geladen. Dann erscheint das Hauptmenü. Euer Bildschirm müßte jetzt folgendermaßen aussehen:

Schirm-Meldung 2

```
Logged drive: A
Active directory:

Work file:
Main file:
Edit   Compile   Run       Save
Dir    Quit      compiler  Options

Text:     0 bytes
Free: 62024 bytes
>
```

Um euch am Anfang nicht mit zuviel Text zu plagen, wollen wir aus diesem Menü nur soviel erklären, wie es für die Arbeit



Das Interesse an dem Turbo-Pascal-Kurs scheint größer zu sein als ich dachte.

jetzt nötig ist. Der Rest kommt später bei konkreten Anwendungen.

Logged drive: LW
Hier wird das aktive Laufwerk (LW) angezeigt. Aktiv heißt, daß auf diesem LW abgespeichert bzw. von diesem LW eingelesen wird. Durch Drücken von „L" (ohne <Enter>) kann das aktive Laufwerk geändert werden. Es erscheint die Zeile „New drive". Ihr gebt die neue Laufwerksbezeichnung ein (A, B, C etc.). Hier noch ein **wichtiger Hinweis** für diejenigen mit CP/M-Rechnern. Wenn ihr im Laufwerk die Diskette wechselt, müßt ihr das extra anmelden, sonst bekommt Ihr später beim Versuch, ein Programm abzuspeichern einen Fehler angezeigt und das Programm ist verloren. Beispiel: Ihr habt nur ein Laufwerk und wollt, nachdem ihr Turbo Pascal eingeladen habt, die Diskette wechseln, um ein Programm zu schreiben und danach abzuspeichern. Diskette im Laufwerk A wechseln; „L" drücken, denselben LW-Buchstaben eingeben, der vorher schon dort stand (also A) und es kann nichts mehr passieren.

Active Directory
Diese Meldung erscheint nur bei MS-DOS Rechnern

Work File:
Work File ist die Datei, mit der aktuell gearbeitet wird (die sich im Speicher des Rechners befindet). Durch Eingabe von „W" kann die Arbeitsdatei geändert werden. Es erscheint die Anzeige „Work file name". Hier kann ein Name eingegeben werden (acht gültige Zeichen als Dateinamen), ein Punkt und drei gültige Zeichen als Dateikennzeichnung. Wird der Punkt und die Dateikennzeichnung weggelassen, wird automatisch ".PAS" angehängt.

Main file:
Dieser Punkt wird später am konkreten Beispiel erläutert.

Edit:
Durch Eingabe von „E" wird der **Editor** aufgerufen. Der Editor ist ein „Textprogramm" mit dem der Programmtext eingegeben wird. Befindet sich noch keine Arbeitsdatei im Speicher, erscheint zunächst die Anzeige „Work file name" und Du mußt einen Dateinamen eingeben. (Der Editor hat übrigens große Ähnlichkeit mit dem Editor von „Wordstar".) Es wäre sehr sinnvoll, wenn ihr im Handbuch von Turbo-Pascal das Kapitel über den Editor einmal durchlest.

Auf „Compile", „Run", „Save", „Dir", „Quit" und „Compile Options" werden wir im konkreten Anwendungsfall zurückkommen.

Grau, teurer Freund, ist alle Theorie: wir wollen jetzt unser erstes PASCAL-Programm schreiben. Programme schreibt man im Editor, deshalb „E" drücken. Es erscheint die Zeile: Work file name: – hier gibst du ein „Zahl" (oder irgendeinen anderen Namen) ein, drückst „Return" und du befindest dich im Editor.
Programme schreibt man in der Rege

TURBO PASCAL – Der Kurs *(Teil 1)*

zeilenweise und deshalb schreiben wir in die erste Zeile

program zahl;
(am Ende jeder Zeile immer die „Return" oder „Enter"-Taste drücken!)
Damit geben wir dem Programm einen Namen. Kümmern Sie sich zunächst nicht um Einzelheiten wie zum Beispiel das Semikolon am Ende der Zeile (obwohl es, wie wir später sehen werden, von entscheidender Bedeutung ist). Der nächste Schritt besteht darin, dem Programm mitzuteilen, daß es mit der eigentlichen Verarbeitung beginnen soll. In diesem ersten einfachen Programm wollen wir lediglich eine Maschine konstruieren, die eine Zahl am Bildschirm ausgibt (hier eine 5). Es wird also zunächst von uns nichts bestimmtes eingegeben. Nun schreiben wir in die nächste Zeile

begin
und in die nächste Zeile den Befehl

writeln (5)
und da nun das Programm bereits zu Ende ist kommt in die nächste und letzte Zeile

end.
(Der Punkt nach „end" ist wichtig, also nicht weglassen).

Die erste Zeile, also die Namensgebung, nennt man den Programmkopf, den Rest den Programmrumpf. Besonders der Programmrumpf sieht in der Regel allerdings wesentlich umfangreicher aus.
Das gesamte Programm hat jetzt folgendes Aussehen:

program zahl;
begin
writeln (5)
end.

Das Programm ist nun fertig und wir verlassen den Editor, um dieses jetzt arbeiten zu lassen. Dazu hält man die „Ctrl"-Taste gedrückt und drückt „K" und anschließend „D". Drücke nun auf die „Blank"-Taste (Wortzwischenraum) und es erscheint das Hauptmenü. Um das Programm laufen zu lassen, muß es zuerst „kompiliert" werden (was das bedeutet, werden wir später noch erklären): dazu drückt man die „C"-Taste. Nach kurzer Zeit erscheint folgende Meldung (bei CPM-Rechnern sieht das etwas anders aus!):
Sollte eine Fehlermeldung erscheinen, hast Du sicher einen Tippfehler gemacht. Korrigiere ihn und compiliere nochmals. Wenn fehlerfrei kompiliert wurde, kann man das Programm laufen lassen: drücke die „R"-Taste (RUN) und es erscheint auf dem Bildschirm die Zahl 5.

Nun kann das Programm noch auf die Diskette gespeichert werden und zwar durch Drücken der Taste „S" (SAVE). Das Programm wird nun auf Diskette gespeichert und kann in der nächsten Lektion wiederverwendet werden. Wenn Du wissen willst, ob es auch wirklich auf Diskette ist, drücke „D" (DIR). Es erscheint die Zeile: Dir mask: – drücke einfach die „Return"-Taste und es erscheint das Inhaltsverzeichnis (DIRECTORY) deiner Diskette. Dort müßte „ZAHL.PAS" stehen bzw. ein anderer Name, den Du dem Programm gegeben hast.

Autoren: Rüdiger Pfister / Roland Schweizer

Schirm-Meldung 3

```
Compiling
4 lines

Code:       0004 paragraphs (    64 bytes), 0D24 paragraphs free
Data:       0002 paragraphs (    32 bytes), OFDA paragraphs free
Stack/Heap: 2BD4 paragraphs (179520 bytes)
```


Weihnachtsgeschenke

Gewinnen Sie fünfmal einen Datenträger Ihrer Wahl!

Was müssen Sie dafür tun? – Ganz einfach!

Wir suchen die besten Vierzeiler:

Zum Beispiel: Die Computronic, ist doch klar, kauf ich auch im nächsten Jahr. Software-Listings, was für Gaben für lediglich 6,– Mark zu haben.

Auf geht's, Dichter und die, die es werden wollen. In jedem steckt ein kleiner Poet! Die Wahl aus dem gut bestückten Sortiment des Tronic-Verlags wird Ihnen schwerfallen!

Hinweis: Vergessen Sie nicht Ihr Computer system mit auf die Karte zu schreiben.

TURBO-PASCAL-KURS: *Teil 2*

Im ersten Teil habt ihr erfahren, was ein Programm ist (zumindest habt ihr einen ersten Eindruck) und ihr habt ein erstes, sehr einfaches Programm kennengelernt. Dieses Programm war natürlich ziemlich unnütz – deshalb werdet ihr in diesem 2. Teil ein weiteres Programm kennenlernen, an dem man schon eher sehen kann, daß Programmieren eine sinnvolle und arbeitsersparende Sache sein kann.

Um dieses Programm verstehen zu können, müssen allerdings einige neue Begriffe erläutert werden. Zunächst wollen wir den Begriff **Variable** uns näher ansehen. Eine Variable in einem Programm hat eine ähnliche Bedeutung wie in der Mathematik: sie steht stellvertretend für irgendeinen konkreten Wert!

Wenn ein Programm z. B. eine Gleichung lösen soll, sagen wir: a + b = c, dann wäre es sehr unpraktisch, für jede spezielle Form dieser Gleichung (z. B. wenn a = 5 und b = 10 ist) ein besonderes Programm schreiben zu müssen. Deshalb existieren in jeder Programmiersprache sogenannte **Variablen**, die man sich bildlich als „Behälter" oder „Schublade" vorstellen kann und die dazu dienen, beliebige „Dinge", nämlich Symbole wie Zahlen oder Buchstaben, aufzubewahren. Um die einzelnen Schubladen, die man zur Verfügung hat, unterscheiden zu können, ist es zweckmäßig, ihnen „Namen" zu geben.

Hat man viele verschiedene Dinge, die man in Schubladen packen muß, ist es außerdem nützlich, bestimmte Schubladen nur für bestimmte Dinge zu reservieren. Z. B. kann man sagen: alle roten Schubladen sind für Zahlen, alle gelben Schubladen sind für Wörter vorgesehen. Fassen wir das oben Gesagte kurz zusammen: Variablen sind eine Art Behälter für Gegenstände, die in einem Programm bearbeitet werden; im 1. Teil haben wir diese Gegenstände **Daten** genannt. Den jeweiligen **Inhalt** einer Variablen nennen wir den **Wert** der Variablen (bearbeitet unser Programm z. B. Zahlen und wir legen in die Schublade mit dem Namen „ALFA" eine „5", dann ist der Wert von ALFA eben 5). Neben einem Wert hat jede Variable natürlich einen **Namen**, diesen Namen kann sich der Programmierer selbst aussuchen und er tut in der Regel gut daran, sich sinnvolle Namen auszusuchen wie ERGEBNIS oder SUMME und nicht einfach Buchstaben wie A, B oder C zu nehmen.

Schließlich ist jede Variable von einem bestimmten **Typ**: damit ist die Art von Objekten gemeint, die man in die jeweilige Schublade legen kann. Was für Typen Variablen in TURBO-PASCAL haben können, wird später genau erklärt. Hier nur ein Beispiel: eine Variable vom Typ INTEGER kann nur ganze Zahlen beinhalten (5, 30, 3345 etc.), während eine Variable vom Typ CHAR nur sogenannte **alphanumerische** Werte beinhalten darf, nämlich Zeichen wie Buchstaben, Ziffern etc.

Eine wesentliche Aufgabe von Programmen besteht darin, daß die Werte (eben die Daten) verändert werden. So können z. B. Zahlen addiert, subtrahiert oder dividiert werden, eben alles, was man aus der Mathematik kennt, kann im Prinzip durchgeführt werden. Eine Folge mathematischer Operationen wie z. B. 5+5–2 kann man in PASCAL prinzipiell genauso schreiben: man hat dann einen **Ausdruck** aufgestellt. Ein Ausdruck ist also eine Zusammenfassung durchführbarer Operationen, beispielsweise algebraischer Operationen. Hat das Programm eine solche Operation durchgeführt, so muß in der Regel das Ergebnis festgehalten werden: wir legen es am besten in eine „Schublade" (Variable) und haben es damit für alle späteren Verwendungszwecke aufbewahrt. Diesen Vorgang, einen bestimmten Wert einer bestimmten Variable zuzuordnen, nennt man **Zuweisung**; in PASCAL dient dazu das Symbol :=(nicht zu verwechseln mit dem Gleichheitszeichen). Eine Zuweisung kann z. B. so aussehen

ERGEBNIS := 5 + 5;

Hier wird im 1. Schritt der **Ausdruck** 5+5 berechnet (das Ergebnis ist 10). Der **Wert** 10 wird im 2. Schritt der Variable mit dem **Namen** ERGEBNIS zugewiesen, diese Variable muß vom **Typ** INTEGER sein, um die ganze Zahl 10 aufnehmen zu können.

Leider wird in anderen Programmiersprachen (z. B. in BASIC) das mathematische Gleichheitszeichen = als Zuweisungszeichen benutzt. Das ist völlig irreführend, denn in einer Zuweisung wird nichts gleichgesetzt, sondern ein Wert in eine Variablenschublade gesteckt. Deshalb darf man sich auch nicht über erlaubte Zuweisungen der Form

SUMME := SUMME + 1;

wundern, die oft sehr nützlich, im Sinne einer mathematischen Gleichung aber natürlich völliger Unsinn sind.

Nach soviel allgemeinem Gerede jetzt eine praktische Aufgabe, die wir mit einem kleinen Programm lösen werden: Die Berechnung des Inhalts einer Kreisfläche.

```
1:  PROGRAM Kreisberechnung_1;
2:  CONST pi = 3.1415926;
3:  VAR radius, kreisflaeche : real;
4:  BEGIN
5:          write ('Radius: ');
6:          readln(radius);
7:          kreisflaeche:=radius * radius
            * pi;
```

„Irgend welche Fragen?"

```
8:        writeln('Der Flaecheninhalt
          ist: ',kreisflaeche);
9: END.
```

Beispielprogramm Kreisberechnung (Version 1)

Dieses kleine Beispielprogramm macht folgendes: Für einen Radius, der vom Benutzer eingegeben wird, wird der Flächeninhalt berechnet und auf dem Schirm ausgegeben. Du kannst das Programm zunächst einmal abtippen und laufen lassen. (Hinweis: Die Zeilennummern gehören nicht zum Programm, sondern dienen dem leichteren Verweisen auf bestimmte Programmstellen: Die Nummern dürfen also nicht eingegeben werden, wenn du das Programm abtippst.)

Damit du aber auch verstehst, was da passiert, soll das Programm jetzt Schritt für Schritt erklärt werden.

Zeile 1: PROGRAM Kreisberechnung_1;

Die erste Zeile ist der **Programmkopf**. Das erste Wort „PROGRAM" (nur mit einem „m") ist ein **reserviertes Wort**. (Reservierte Wörter sind Bestandteil von Turbo-Pascal und können nicht vom Benutzer neu definiert werden. Reservierte Wörter werden von uns immer großgeschrieben. Das heißt aber nicht, daß du sie auch groß schreiben mußt, wenn du Dein Pascal-Programm schreibst.)

„PROGRAM" leitet ein Pascal-Programm ein, bedeutet also, daß das Programm hier beginnt. Hinter „PROGRAM" muß der Name des Programms stehen. Dieser Name (der Fachbegriff hierfür lautet: **Bezeichner**, und zwar ein selbstgewählter) darf aus einer beliebigen Kombination von Buchstaben, Zahlen oder Unterstreichungszeichen bestehen. Wichtig ist nur, daß zu Beginn des Namens keine Ziffer stehen darf und daß die Länge des Namens durch die Zeilenlänge (max. 127 Zeichen) begrenzt wird. Nach dem Namen muß dann ein Semikolon (;) stehen. (Anmerkung: Bei Turbo Pascal muß die Kopfzeile nicht unbedingt eingetragen werden, d. h., das Programm läuft auch ohne. Empfehlenswert ist diese „Namensgebung" jedoch auf jeden Fall.)

Das Semikolon

Zum Semikolon möchten wir eine grundsätzliche Anmerkung machen, weil es für ein Pascal-Programm ziemlich bedeutsam ist. Das Semikolon hat die grundlegende Aufgabe **zu trennen**. Es trennt den Programmkopf vom übrigen Programm. Ihr werdet weiter unten sehen, daß es im **Deklarationsteil** einzelne Deklarationen trennt. Es trennt auch einzelne **Anweisungen** voneinander (s. u.). (Für BASIC-Kenner: das, was in BASIC die Return-Taste oder der Doppelpunkt ist, ist für Pascal das Semikolon). Am Anfang werdet ihr wahrscheinlich des öfteren das Semikolon mal vergessen und ihr werdet dann feststellen, wie verschiedenartig die Fehlermeldungen des Compilers ausfallen können, welche Bedeutung also das Semikolon für ein Pascal-Programm hat.

Zeile 2 und 3 enthält den sogenannten **Deklarationsteil**. In diesem Teil wird alles deklariert (oder anders ausgedrückt: definiert), was im späteren Hauptprogramm (z. B. an Variablen) benötigt wird.

Zeile 2: CONST pi = 3.1415926;

Hier wird mit dem reservierten Wort „CONST" eine Konstante mit dem Namen „pi" definiert. Der Wert von pi soll 3,1415926 sein. (Das Dezimalkomma muß als Punkt eingegeben werden.) Die Konstante „pi" muß im gesamten Programm den gleichen Wert behalten (das sagt ja schon der Name) und darf nicht verändert werden.

Regel: Nach „CONST" kommt der Name der Konstanten (ein Bezeichner, s. o.), danach folgt ein „="-Zeichen und dann der Wert der Konstanten. Als Abschluß das Semikolon. (CONST Name = Wert;)

Zeile 3: VAR radius kreisflaeche : real;

Mit dem reservierten Wort „VAR" beginnt die Definition von Variablen. Definiert werden muß der **Typ**, den die Variable haben soll (die **Werte** selber sind ja verschieden, eben variabel). Abgesehen davon, daß man in Pascal selbst Datentypen definieren kann (doch das kommt später), stellt Turbo-Pascal fünf Datentypen zur Verfügung:

integer, real, boolean, char und byte.

Die ersten zwei Typen und der letzte Typ sind Zahlen-Typen. **Integer** bedeutet eine ganze Zahl (Bereich -32768 bis +32767), **real** bedeutet eine reelle Zahl oder anders ausgedrückt eine Dezimalzahl (Bereich: 1E-38 bis 1E+38; diese Schreibweise wird euch wahrscheinlich verwundern, aber so werden in Pascal Zahlen in Potenz-Form geschrieben. Du kennst diese Zahlen besser in dieser Form: 10-38, bzw. 10+38, also eine 1 mit 38 Nullen, einmal negativ, einmal positiv). **Boolean** ist ein sogenannter „Wahrheitstyp" mit den Werten „true" (für wahr) und „false" (für falsch). **Char** bedeutet ein **alphanumerisches** Zeichen (also das Alphabet oder Ziffern). **Byte** ist ein Teilbereich von „integer" (Bereich: 0 bis 255). (Wir werden alle Datentypen im Zusammenhang konkreter Anwendungen noch genauer besprechen.)

Der Datentyp für unsere Variablen „radius" und „kreisflaeche" ist also „real". Das erklärt sich so, daß beide Variablen auf jeden Fall Zahlen sind, aber der Bereich „integer" nicht in Frage kommt, weil es sonst nicht möglich wäre, einen Radius von z. B. 2,50 einzugeben. Das Programm würde mit einem Fehler abbrechen. (Ihr könnt das ja mal ausprobieren. Dabei kann nichts passieren.) Ist der Datentyp einer Variablen definiert, so ist er für das gesamte Programm, in dem diese Variable benutzt wird, festgelegt. Dabei ist folgendes zu beachten: Ist eine Variable als Real-Typ definiert, so darf trotzdem eine Integer-Zahl (also eine ganze Zahl) benutzt werden. Umgekehrt geht das jedoch nicht.

Regel: Nach „VAR" kommt der Name der Variablen. Haben mehrere Variablen den gleichen Datentyp, werden sie durch ein Komma getrennt. Nach der Variablen kommt ein Doppelpunkt und dann der Datentyp. (VAR variable1,variable2,variable 3 : Datentyp;)

Die Reihenfolge der Deklarationsteile ist übrigens bei Turbo-Pascal (im Gegensatz zu Standard-Pascal) beliebig. Wir halten uns jedoch zukünftig immer an die Reihenfolge von Standard-Pascal. Dies macht es einfacher, eventuell mal in einer anderen Pascal-Version zu programmieren.

In Zeile 4 bis Zeile 9 befindet sich der Anweisungsteil unseres Programms. Hier wird die eigentliche Arbeit des Programms geleistet. In diesem kleinen Programm könnt ihr übrigens das klassische **EVA-Prinzip** (**E**ingabe - **V**erarbeitung - **A**usgabe) sehen, das in fast jedem Programm vorkommt.

Zeile 4: BEGIN

Das reservierte Wort „BEGIN" leitet den Anweisungsteil ein.

Zeile 5: write('Radius: ');

Write ist eine Prozedur, die, ganz allgemein gesagt, zur Ausgabe dient. In diesem Fall bewirkt sie, daß der in Klammern stehende und in Anführungszeichen stehende Text "Radius: " auf der Bildschirm ausgegeben wird. Das, was ausgegeben werden soll, muß immer in Klammern hinter „write" stehen.

Zeile 6: readln(radius);

Die **read-Prozedur** ist das Gegenstück zu „write"-Prozedur und ermöglicht, ganz

allgemein gesagt, die Eingabe von Daten in ein Programm. Das **readln**, das wir benutzt haben, unterscheidet sich von **read** nur dadurch, daß bei readln (ln = line) der Cursor in die nächste Zeile springt. Durch readln (bzw. read) wird der Programmablauf unterbrochen und der Benutzer bekommt die Möglichkeit, über die Tastatur eine Zahl einzugeben. Das Ende der Eingabe wird durch Drücken der Return-Taste erkannt und die eingegebene Zahl wird der Variablen „radius" zugeordnet. „Radius" hat jetzt den Wert, der eingegeben wurde (z. B. 6.50). Das was ausgegeben werden soll, muß immer in einer Klammer hinter **readln** bzw. hinter **read** stehen.

Zeile 7: kreisflaeche:= radius * radius * pi;

In dieser Zeile wird gerechnet (die Sternchen sind das Multiplikationszeichen). Das, was in dieser Zeile steht, nennt man eine **Zuweisung** (auch wenn die Zeile einer mathematischen Gleichung ähnlich aussieht, ist es keine). Die Zeile wird folgendermaßen bearbeitet: Zuerst wird der rechte Teil der Anweisung ausgewertet. In unserm speziellen Fall bedeutet das, daß der folgende „Ausdruck" berechnet wird (Bsp. für Radius: 6.50): 6.50 * 6.50 * 3.1415926. Im zweiten Schritt wird dann das Ergebnis dieser Rechnung (132.73228735) der Variablen „kreisflaeche" **zugewiesen**. Das Zeichen „:=" nennt man **Zuweisungssymbol**. Jetzt enthält die Variable „kreisflaeche" den Wert 132,73228.

Zeile 8: writeln('Der Flaecheninhalt ist: 'kreisflaeche);

Writeln unterscheidet sich von der Prozedur „write" nur dadurch, daß der Cursor in die nächste Zeile bewegt wird. Interessanter ist, was in der Klammer ist. Zunächst steht da ein Text in Anführungszeichen: "Der Flaecheninhalt ist: ". Dieser Text wird so auf dem Bildschirm ausgegeben. Danach folgt ein Komma. Dieses Komma wirkt als **Trenner** zwischen verschiedenen Teilen (ganz analog zur Variablendefinition in Zeile 3), die ausgegeben werden sollen. Nach dem Komma kann jetzt etwas Neues aufgeführt werden, in unserem Fall die Variable „kreisflaeche". Dadurch, daß „kreisflaeche" ohne Anführungszeichen geschrieben wurde, wird jetzt nicht das Wort „kreisflaeche" auf den Bildschirm geschrieben, sondern „kreisflaeche" wird als Variable erkannt und der Inhalt der Variablen wird auf dem Schirm ausgegeben. (Anmerkung: identisch zur Zeile 8 wäre die Programmzeile:
write('Der Flaecheninhalt ist: ');
writeln(kreisflaeche);

Zeile 9: END.

Das **END** beendet den Anweisungsteil. Der Punkt ist sehr wichtig, weil er, nach dem „END" stehend, das Programmende anzeigt.
So, das war das Programm. Wenn ihr es jetzt eingegeben habt, geht aus dem Editor heraus (Ctrl-K-D), speichert das Programm ab (mit „s") und laßt es mal laufen (mit „R"). Zunächst wird es automatisch compiliert. Sollte der Compiler einen Fehler melden, habt ihr wahrscheinlich einen Tippfehler begangen.
Wenn ihr das Programm einmal laufen laßt, dann seid ihr vielleicht mit uns der Meinung, daß es ein paar Schönheitsfehler hat. Zum einen steht über dem Radius noch „Running" und der ganze Text der Compilation. Zum zweiten sieht das Ergebnis reichlich merkwürdig aus. Bei dem Ergebnis hat das damit zu tun, daß „Pascal" jede „real"-Zahl (Dezimalzahl) in Potenzschreibweise mit 11 Stellen ausgibt. Diese zwei Schönheitsfehler lassen sich jedoch relativ einfach beheben.
Geht zunächst wieder in den Editor (mit „E"). Ändert Euer Programm jetzt so ab, daß es folgendermaßen aussieht:

```
 1: PROGRAM Kreisberechnung_2;
 2: CONST pi = 3.1415926;
 3: VAR radius, kreisflaeche : real;
 4: BEGIN
 5:         clrscr;
 6:         write('Radius: ');
 7:         readln(radius);
 8:         kreisflaeche:= radius *
            radius * pi;
 9:         writeln('Der Flaecheninhalt ist: ');
10:         writeln(kreisflaeche:10:2);
11: END.
```

Denkt beim Ändern an die Editorbefehle Ctrl-N (für Zeile einfügen), Ctrl-Y (für Zeile löschen), Ctrl-G (für Zeilchen löschen) bzw. eine entsprechende Lösch (-Delete)-Taste bei eurem Rechner.
Folgendes hat sich geändert:
Zeile 5: clrscr;
Diese „clrscr" ist eine Kurzform für „Clear Screen" (Bildschirm löschen) und ist eine sogenannte **Standardprozedur**, die von Turbo-Pascal zur Verfügung steht. Sie bewirkt folgendes: Sie löscht den Bildschirm und plaziert den Cursor in die linke obere Ecke.
Zeile 10: writeln(kreisflaeche:10:2);
Neu ist hier, daß hinter der Variable „kreisflaeche" folgendes steht: „:10:2"
:10 bedeutet:
Die Zahl, die für Kreisfläche berechnet wurde, wird rechtsbündig so ausgegeben, daß die letzte Ziffer in der 10. Spalte steht. Es werden also links entsprechend viele Leerzeichen eingefügt.
:2 bedeutet:
Die Zahl (die vom Typ „real" sein muß) wird mit 2 Nachkommastellen ausgegeben.
Wenn ihr das Programm geändert habt, dann laßt es nochmal laufen. Jetzt sieht das doch schon viel besser aus.
Das war's für heute. Das nächste Mal werden wir uns mit Kontrollstrukturen beschäftigen.

IMPRESSUM

Computronic
ISSN 0179-6739
erscheint alle 2 Monate im Tronic-Verlag, Postfach 870
3440 Eschwege, Tel. 0 56 51 / 3 00 11

Herausgeber: Axel Credé
Chefredakteur: Thomas Brandt
Redaktion: Ottfried Schmidt, Frank Brall, Hartmut Wendt, Manfred Kleimann, Bernd Zimmermann, Martina Strack, Uwe Knierim

Programmautoren: Roland Mayer, Frank Gehring, Frank Wittenmayer, Gerhard Plietsch, Karsten Bogusch, Rudolf Neumann, Uwe Schäfer

Gesamtherstellung:
Druckhaus Dierichs Kassel, Frankfurter Str. 168, 3500 Kassel

Vertrieb:
Inland (Groß-, Einzel- und Bahnhofsbuchhandel) sowie Österreich und Schweiz:
Verlagsunion
Friedrich-Bergius-Straße 20
6200 Wiesbaden
Telefon 0 61 21 / 26 60

Anzeigenverwaltung:
Hartmut Wendt, Tronic-Verlag, Postfach 870, 3440 Eschwege, Tel. 0 56 51 / 3 00 11

Mitglied der Informationsgemeinschaft zur Festellung der Verbreitung von Werbeträgern e.V. (IVW, Bad Godesberg)

Postfach 870 (Postbox 870),
D-3440 Eschwege, Tel. 0 56 51 / 3 00 11

Erscheinungsweise: Erstverkaufstag von COMPUTRONIC ist Anfang des Monats

Anzeigenpreisliste:
Bitte Mediaunterlagen anfordern.

Software-Service und Vertrieb: Annelie Kratzenberg, Heike Rabe.

Urheberrecht:
Alle in COMPUTRONIC veröffentlichten Beiträge sind urheberrechtlich geschützt. Alle Rechte, auch Übersetzungen, vorbehaltungen.
Reproduktionen jeder Art (Fotokopien, Microfilm, Erfassung in Datenverarbeitungsanlagen usw.) bedürfen der schriftlichen Genehmigung des Verlags. Alle veröffentlichte Software wurrde von Mitarbeitern des Verlages oder von freien Mitarbeitern erstellt.
Aus Ihrer Veröffentlichung kann nicht geschlossen werden, daß die beschriebenen Lösungen oder Bezeichnungen frei von Schutzrechten sind.

Bezugspreise:
Einzelheft 6,50 DM

Autoren, Manuskripte:
Der Verlag nimmt Manuskripte und Software zur Veröffentlichung gerne entgegen. Sollte keine andere Vereinbarung getroffen sein, so gehen wir davon aus, daß Sie mit einem Honorar von 120,– DM pro abgedruckter Seite im Heft einverstanden sind.

Titelbild: Fischer, Bad Hersfeld

TURBO-PASCAL-KURS Teil 3

Kontrollstrukturen

Wie schon angekündigt wollen wir uns heute mit Kontrollstrukturen befassen. Kontrollstrukturen haben die Aufgabe, den Programmablauf zu kontrollieren oder anders ausgedrückt, die Ausführung von Anweisungen zu steuern.

Bisher stand ich ja mit Computern auf Kriegsfuß, aber der Turbo-Pascal-Kurs in der „Computronic" ruft neue Gefühle in mir hervor!

Bisher schien es so, als müsse der Ablauf eines Programmes streng Sequentiell sein, d.h. es werden alle Zeilen des Programms in genau der Reihenfolge abgear beitet und die entsprechenden Befehle ausgeführt, wie Sie vom Programmierer geschrieben wurden. Mit Hilfe bestimmter Befehle kann man erreichen, daß der sequentielle Ablauf verlassen wird. Die sogenannten Kontrollstrukturen dienen dazu, das Programm flexibel zu gestalten. Ein Programm - wir haben darauf schon öfter hingewiesen- soll ja für viele verschiedene Aufgaben nützlich sein (z.B. soll ein Additionsprogramm alle möglichen Additionen ausführen können) und nicht nur für eine spezielle. Deshalb ist es dringend notwendig, je nach Art der Aufgabe eine Möglichkeit vorzusehen, damit das Programm entsprechend reagieren kann.

Dazu ein konkretes Beispiel: Schaut euch einmal das Programm „Kreisberechnung" aus der letzten Folge an. Das Programm berechnet den Flächeninhalt eines Kreises für einen eingegebenen Radius. Jedem ist wohl klar, daß der Radius eines Kreises keine negative Zahl sein darf. Wenn Ihr aber dem Programm z.B. - 2 für den Radius eingebt, bekommt Ihr als Ergebnis 12.57 (-2*-2 *Pi=4*Pi=12.57). Das heißt, man müßte hier eigentlich die Eingaben auf Zulässigkeit prüfen und negative Zahlen nicht zulassen.

Die bedingte Anweisung (if ... then ... else)

Wir wollen das an einem anderen Beispiel realisieren. Aufgabe: Ein Programm soll für eine eingegebene Zahl die Quadratwurzel ausgeben. Dabei sind negative Zahlen nicht zugelassen. (Der Rechner würde in diesem Fall sogar das Programm mit einem Fehler abbrechen.) Das Programm dazu sieht folgendermaßen aus: (Listing 1)

Bis zur Programmzeile 8 dürfte alles bekannt sein. Anderenfalls lest dies dann bitte noch einmal in Kursteil 1 und 2 nach.

In den Zeilen 9 bis 11 steht die if-Anweisung, die die Eingabe von negativen Zahlen verhindern soll. Die Sruktur dieser Anweisung besitzt folgendes Aussehen: IF Bedingung THEN Anweisung1 (wenn Bedingung erfüllt) ELSE Anweisung2 (wenn Bedingung nicht erfüllt)

(Hinweis: vor dem „ELSE" darf kein Strichpunkt stehen. Dieses if-then-else wird als Einheit betrachtet).

Die Bedingung in Zeile 9 lautet; die eingegebene Zahl muß kleiner als Null sein. Ist diese Bedingung erfüllt (z.B. für -2) dann wird die Anweisung hinter THEN in Zeile 10 ausgeführt. Ist die Bedingung nicht erfüllt (z.B. für 2) dann wird die Anweisung hinter ELSE in Zeile 11 ausgeführt. Zunächst wird Euch vielleicht erstaunen, daß nach dem ELSE ein BEGIN steht, doch dazu später. Zunächst schauen wir uns die Anweisung in Zeile 12 an: „wurzel:-sqrt(zahl);". Anfangs haben wir es mit einer Zuweisung zu tun (siehe Folge 2). Der rechte Teil wird der Variablen „wurzel" zugewiesen. Rechts steht „sqrt(zahl)". Dieses „sqrt" ist eine Standardfunktion, die von Turbo-Pascal zur Verfügung gestellt wird. Diese berechnet die Wurzel der Zahl, die in Klammern dahinter steht, also sqrt (4) gibt als Ergebnis die Wurzel von 4 an., also 2, zurück. Dieses Ergebnis wird in unserem Fall der Variablen „wurzel" zugewiesen. Die Anweisung in Zeile 13 müßte bekannt sein (siehe Folge 2).

Nun zum BEGIN nach dem ELSE in Zeile 11. In Folge 2 haben wir schon einmal vom Anweisungsteil gesprochen, der nach dem Deklarationsteil kommt. Eine Anweisung ist z.B. „write ('Gib eine Zahl ein: ');" oder „kreisfläche:=radius*radius*pi;". Eine Anweisung veranlaßt den Rechner ständig etwas zu tun und jede einzelne Anweisung muß mit einem Strichpunkt abgeschlossen werden.

Bei dem Kontrollelement „if-then-else" gilt zunächst, daß sowohl hinter THEN als auch hinter ELSE jeweils eine Anweisung stehen darf. Bei uns ist das nach THEN der Fall. Nach ELSE sollen jedoch zwei Anweisungen ausgeführt werden, die zusammen gehören, was auch als Verbundanweisung bezeichnet wird. Eine Verbundanweisung beginnt immer mit BEGIN und endet mit END.

Nehmen wir mal an in unserem Programm stünde das BEGIN nach ELSE nicht: (Listing 2)

Der Programmablauf wäre jetzt folgendermaßen:

1. Fall: Die Zahl ist kleiner als Null. Zunächst würde die Anweisung nach THEN ausgeführt (Zeile 10). Die eine Anweisung nach ELSE (Zeile 12) würde nicht ausgeführt. Danach würde das Programm weitergehen und zwar bei Zeile 13.

2. Fall: Die Zahl ist größer oder gleich Null. Zunächst würde die eine Anweisung nach ELSE (Zeile 12) ausgeführt. Danach würde das Programm weitergehen bei Zeile 13.

Im ersten Fall gäbe es also einen Fehler im Programmablauf, im zweiten Fall nicht.

Verschachtelung der bedingten Anweisung

Die IF-THEN-ELSE-Anweisung kann man auch verschachteln, d. h. innerhalb

einer IF-THEN-ELSE-Anweisung steht wieder eine bedingte Anweisung. Das sieht dann folgendermaßen aus: (Listing 3)

Von Interesse sind nur die Zahlen 9 bis 12, alles andere bleibt unverändert. Der Ablauf ist folgendermaßen: In Zeile 9 wird geprüft, ob die Zahl kleiner als Null ist.
Fall 1: Zahl ist kleiner als Null. Zeile 10 wird ausgeführt. Danach weiter zu Zeile 17 (Programmende).
Fall 2: Zahl ist nicht kleiner als Null. Es geht zu Zeile 11. Hier wird geprüft, ob die Zahl Null ist.
Fall 2.1: Die Zahl ist Null. Da die Wurzel aus Null immer Null ist, braucht nicht gerechnet zu werden. Es wird die Zahl 12 ausgeführt. Danach geht es weiter zu Zeile 17.
Fall 2.2: Die Zahl ist größer als Null. Die Verbundanweisung nach ELSE wird ausgeführt (Zeile 13). Danach geht es weiter zu Zeile 17.
Vielleicht ist auch schon klar, warum man mit solchen Verschachtelungen vorsichtig sein soll. Sie haben die Angewohnheit sehr schnell sehr unübersichtlich zu werden.
Die bedingte Anweisung kann auch ohne ELSE benutzt werden. Vielleicht ist Euch aufgefallen, daß bei der bedingten Anweisung jeweils zwischen zwei Fällen unterschieden wurde. Man nennt dies deshalb auch eine zweiseitige Auswahl (wird der ELSE-Teil weggelassen, handelt es sich um eine einseitige Auswahl).

Die Case-Anweisung

Mit der CASE-Anweisung kann auf sehr einfache und übersichtliche Art eine mehrseitige Auswahl getroffen werden. Wir wollen das wieder an einem Beispiel zeigen. Bei dem nachfolgenden Programm handelt es sich um ein kleines Rechenprogramm, das folgendes leisten soll: Man kann aus den vier Grundrechenarten eine aussuchen, danach zwei Zahlen eingeben, die dann entsprechend berechnet werden, als Ausgabe erscheint dann das Ergebnis. Da man einen Ausdruck (z.B. „4 +6“) nicht direkt in den Computer zur sofortigen Berechnung eingeben kann, muß diese Eingabe zerlegt werden. In unserem Beispiel sind „4“ und „6“ Zahlen und von daher nicht problematisch. Das Rechenzeichen (z.B. „+“) ist jedoch bei der Eingabe vom Datentyp char (siehe Folge 2) und kann erst rechenintern zu einem Teil eines Ausdrucks „erhoben“ werden, der anschließend berechnet werden kann.
Beispielprogramm für die CASE-Anweisung: (Listing 4)

Wir wollen uns das Programm genauer ansehen. Neu ist die Zeile 3. Hier wird eine Variable „rechenzeichen“ definiert und zwar als Datentyp char (s.o.). Das VAR muß in diesem Fall nicht wiederholt werden.
In Zeile 12 bis Zeile 17 steht die CASE-Anweisung. Sie ist folgendermaßen zu verstehen: Wenn die Variable „rechenzeichen“ als Inhalt (CASE rechenzeichen OF) „+“ hat (Zeile 13), dann führe die nach dem Doppelpunkt stehende Anweisung aus (hier können auch zwischen BEGIN und END eingeschlossen mehrere Anweisungen stehen). Wenn der Inhalt der Variablen „rechenzeichen“ z.B. ein „*“ ist, dann wird die Zeile 15 ausgeführt usw.. Enthält die Variable „rechenzeichen“ keines der vier aufgeführten Zeichen, dann geschieht in der CASE-Anweisung nichts. In unserem Fall bedeutet das jedoch, daß ein unzulässiges Rechenzeichen eingegeben wurde. Turbo-Pascal bietet hier die Möglichkeit, solch eine Fehleingabe aufzufangen und zwar mit dem schon bekannten reservierten Wort ELSE. Fügt zwischen Zeile 16 und 17 folgendes ein (denkt an CTRLN für „Zeile einfügen“):

ELSE writeln ('Rechenzeichen falsch! Ergebnis deshalb falsch!! ');

Damit ist der Benutzer gewarnt, denn die Ausgabe des Ergebnisses kann in unserem Programm nicht verhindert werden (solche Möglichkeiten werden sich Ihnen später eröffnen). Euer CASE-Teil muß dann so aussehen: (Listing 5)

Jede CASE-Anweisung muß mit einem END abgeschlossen werden.

★ ☆ ★ ☆ ★ ☆ ★ ☆ ★ ☆ ★

Wieder ein Wettbewerb

Holt Euch die Preise – auf diese Art und Weise:

B	L	H	O	M	E	C	O	M	P	U	T	E	R	F	I
E	R	U	P	F	L	I	D	G	Z	N	H	D	Y	B	G
Z	S	N	G	C	Y	P	W	L	O	F	I	P	U	F	H
N	O	I	T	K	A	D	E	R	B	S	Q	W	Z	T	A
K	W	Z	E	B	I	Z	T	X	K	M	G	N	H	R	W
M	C	L	R	S	E	U	K	E	W	F	K	U	L	O	M
A	T	C	O	M	P	U	T	R	O	N	I	C	M	N	K
H	V	P	Y	M	Z	T	Q	L	N	T	I	O	P	I	Q
R	I	B	O	U	E	Y	H	A	D	M	E	A	R	C	F
V	K	C	E	I	G	F	B	O	Y	U	R	Z	H	B	L

Vor Euch seht Ihr so ein verwirrendes Kästchen mit ganz vielen Buchstaben drin. Ob Ihr es glaubt oder nicht, in diesem Buchstaben-Chaos stehen folgende Wörter drin:

TRONIC
COMPUTRONIC
HOMECOMPUTER
REDAKTION
DISKETTE

Diese Wörter können nun horizontal, vertikal oder diagonal, vorwärts oder rückwärts in dem Kästchen stehen. Viel Spaß beim Suchen, denn es gibt wieder fünf Programm-Disketten / Kassetten zu gewinnen!

TURBO-PASCAL ★ TURBO-PASCAL

Komfortable Eingabe-Routinen für TURBO-PASCAL

Als Unterstützung unseres TURBO-PASCAL-Kurses haben wir uns diesmal drei sehr nützliche Utilities ausgedacht.

Eine große Schwachstelle von Pascal-Systemen ist die Eingabe von Real-/ Integer-Werten. Insbesondere bei professionellen oder umfangreichen Anwendungen, welche eine abbruchsichere Eingabe verlangen, sind die vorhandenen PASCAL-PROCEDUREN nur selten ausreichend. Das nachfolgende Pascal-Listing enthält drei neue Eingabefunktionen, die nun diesem Problem endlich ein Ende bereiten.

Da in Pascal eigentlich nur die Eingabe von STRINGS, INTEGER- und REAL-Werten Schwierigkeiten bereiten, wurde für jeden Variablentyp eine eigene Eingabe-Funktion entwickelt. Die Funktionen erhielten die sinnvolle Bezeichnung GETSTR, GETINT und GETREAL. Je nach Anwendung hat man die Wahl, entweder eine einzelne oder alle Routinen in sein Programm mit einzubinden. In unserem abschließenden Demoprogramm haben wir beispielweise alle Funktionen mit eingebunden. Zu beachten ist, daß vor oder mit der Einbindung die Variable CURTASTE als Character definiert wird. Außerdem muß ein Variablentyp mit der Bezeichnung STRING80 als ein 80 Zeichen umfassender String definiert werden, da Funktionen bekanntlich eine direkte STRING-Zuweisung nicht erlauben. Das abgedruckte Programm wurde auf einem IBM-PC unter Turbo-Pascal entwikkelt, läßt sich jedoch durch geringfügige Änderungen der Tastaturcodes auch auf andere Rechner oder Pascalsysteme übertragen.

Die neue SYNTAX

Die Syntax der einzelnen Funktionen ist relativ einfach, jedoch recht umfangreich:

```
xxxx := GETSTR  (x,y,l,v,a,b,f,g,L)
xxxx := GETINT  (x,y,l,v,a,b,min,max,L)
xxxx := GETREAL
        (x,y,l,k,v,a,b,min,max,L)
```

Die Werte x und y stehen für die Bildschirmposition der Eingabeanweisung. Unter Eingabeanweisung versteht man einen Hinweis auf die verlangte Eingabe, ähnlich dem „INPUT-Text" in der Programmiersprache Basic. Die Werte x und y entsprechen der Pascal-GOTOXY-Funktion, beeinflussen jedoch nicht die augenblickliche Position des Cursors.

Der Wert 1 gibt die maximale Länge der Eingabe an. Um eine übersichtliche und gleichermaßen komfortable Eingabe zu ermöglichen, wird das entsprechende Eingabefeld invers dargestellt, bzw. durch eine andere Hintergrundfarbe gekennzeichnet. Das gekennzeichnete Feld der Eingabe kann später nur über die Taste ENTER verlassen werden und ist somit abbruchsicher. Bei der Funktion GETREAL gibt l die Gesamtlänge des Eingabefeldes an, inclusive Nachkommastellen und der eigentlichen Kommastelle.

Die Variable v ist vom gleichen Typ wie die eigentliche Eingabe-Funktion. Sie kann dazu benutzt werden, um dem Eingabefeld bereits einen Wert oder einen String zuzuordnen. Dieser kann dann nach dem Aufruf der Funktion editiert oder überschrieben werden. Das Demoprogramm veranschaulicht den Vorgang sehr deutlich.

Aus Ihnen wird nie etwas, wenn Sie sich nicht mit Turbo-Pascal befassen!

In der Variable a kann ein Eingabehinweis, welcher durch x und y positioniert wird, vorgegeben werden. Möchte man stattdessen eine Maske verwenden, so kann die hier auch ein Leerstring angegeben werden.

Die Variable b ist vom Typ Byte und dient lediglich als Flag. Enthält dieses Flag den Wert 0, so kann die Eingabe lediglich durch die Taste ENTER beendet werden. Enthält diese Flag jedoch den Wert 1, so kann Eingabe auch über die Cursortasten HOCH und RUNTER abgeschlossen werden. In diesem Fall wird der Variablen CURTASTE der Tastencode 72 für HOCH oder 80 für RUNTER zugeordnet. Diese Option wurde vorgesehen, um komfortable Masken zu gestalten, in denen Felder zurück oder vorgesprungen werden dürfen. Das Demoprogramm zeigt dafür eine schönes Beispiel.

Bei der Funktion GETSTR lassen sich über die Variable f vom Typ Byte weitere Optionen einstellen. Der Wert 0 veranlaßt, daß die Eingabe so übernommen wird, wie sie am Bildschirm eingegeben wurde. Der Wert 1 verursacht ein Abschneiden der vorderen Leerzeichen. Durch den Wert 2 werden die hinteren Leerzeichen abgeschnitten. Der Wert 3 schneidet die vorderen und hinteren Leerzeichen ab.

Auch die Variable g dient als Flag vom Typ Byte. Enthält sie den Wert 0, so wird der String nicht verändert. Enthält sie jedoch den Wert 1, so wird die Eingabe in Großbuchstaben umgewandelt. Dies eignet sich insbesondere für STRING-Abfragen und Vergleiche.

Die Variable L dient ebenfalls als Flag vom Typ Byte. Der Wert 0 steht für Standardeingabe. Der Wert 1 sorgt dafür, daß die angezeigte Feldbeschreibung, sowie das Feld selbst, nach der Eingabe wieder gelöscht wird. Dies ist beispielsweise bei Programmen sinnvoll, welche verschiedene Eingaben an der gleichen Position erwarten.

Die Funktionen GETINT und GETREAL verlangen neben den Standardparametern auch die Angabe der Variablen min und max. Diese beiden Variablen geben an, in welcher Grenze sich die einzugebende Zahl befinden muß. Die Eingabe kann nur dann beendet werden, wenn sich die Zahl innerhalb dieser Grenzen befindet.

Die Funktion GETREAL erlaubt, über die Varaible k zu entscheiden, wieviel Nachkommastellen jeweils in dem entsprechendem Eingabefeld dargestellt werden dürfen. Auch diese Option wird sehr gut aus dem Demoprogramm ersichtlich. Damit haben wir alle erforderlichen Parameter besprochen. Es liegt nun in der Hand des Anwenders, wie er die Funktionen verwendet. Grundsätzlich kann man sagen, daß durch die Vielzahl der Optionen sehr viele Eingabestrukturen denkbar sind. Selbst Masken sind relativ leicht zu bewerkstelligen, was durch das Demo demonstriert wird. Der nächste Schritt wäre ein Maskengenerator, vieleicht findet hier der eine oder andere Leser eine entsprechende Lösung. Es sei noch abschließend gesagt, daß die Eingabefelder absolut abbruchsicher arbeiten und mit den Cursortasten, sowie beiden DEL-Tasten wie üblich editiert werden können.

Übrigens: PC-Besitzer können das Programm für 20 DM auch auf einer Diskette im MS-DOS-Format beziehen. Die Bestellnummer lautet PC-D1.

Das Listing ➡

TURBO-PASCAL ★ TURBO-PASCAL

●●● Teil 1 ●●●

```
type getfeld.inc
{ Komfortable Eingabe von STRINGS, REAL- und INTEGER-ZAHLEN }
{ Neue Funktionen sind: GETSTR / GETREAL / GETINT }
{ (c) F.B. exclusive fuer den TURBO-PASCAL Kurs COMPUTRONIC' }

type
  string80 = string[80];

var
  curtaste:char;  (enthaelt letzten Tastencode)
                  (und erlaubt somit den Aufbau komfortabler Masken)

{SYNTAX:        Funktion GETSTR (x,y,l,V,A,B,F,G,L)
 x,y = Position der Eingabebezeichnung (z.B. 'Name:')
 l   = Maximale Laenge der Eingabe (max 80 Zeichen);
 V   = Stringvariable enthaelt ursprueglichen Inhalt
 A   = String welcher die Eingabe kennzeichnet (z.B. 'NAME:' oder '')
 B   = Abbruch wenn Cursor nach oben o. unten; CURTASTE enthaelt Tastencode
 F   = 1 Leerzeichen am Ende abschneiden
       2 Leerzeichen am Anfang abschneiden
       3 Leerzeichen am Anfang und Ende abschneiden
       0 Standard Eingabe
 G   = 1 In Grossbuchstaben wandeln
       0 Standard Eingabe
 L   = 1 Feld und Feldbeschreibung nach der Eingabe loeschen
       0 Standard Eingabe  }


function getstr (x,y,laenge      :byte;
                 Ursprung,Anzeige:string80;
                 Abbruch,Format  :byte;
                 Gross,Loeschen  :byte   ):string80;
const
  Reversecolor = LightMagenta;

Var
  Puffer :string80;
  altx,alty,cx,mx:byte;
  i,t:integer;  (hilfsvariablen)
  taste,taste2:char;
begin
  altx:=wherex;
  alty:=wherey;
  mx:=x;
  taste2:=#0;
  Puffer:=copy(Ursprung,1,laenge);
  i:=length(Puffer);
  if i<laenge then begin
    i:=laenge-i;
    for t:=1 to i do Puffer:=Puffer+' ';
  end;
  gotoxy (x,y);
  write (Anzeige);
  x:=wherex;
  y:=wherey;
  cx:=x;                          (cursor x-position)
  textbackground(Reversecolor);   (bei CPC LOWVIDEO)
  write (Puffer);
  repeat
    if cx=(x+laenge) then begin
      write(^g);
      cx:=cx-1;
    end;
    gotoxy (cx,y);
    read (kbd,taste);
    curtaste:=taste;
    if (taste=#27) and keypressed then begin
      read (kbd,taste);
      curtaste:=taste;
      taste2:=taste;
      if (taste=#75) and (cx>x) then cx:=cx-1;
      if (taste=#77) and (cx<(x+laenge-1)) then cx:=cx+1;
      if (taste=#83)  then begin (DEL unten)
        for i:= 1 to laenge do puffer[i]:=' ';
        cx:=x;
        gotoxy (cx,y);
        write (puffer);
      end;

      if abbruch=0 then taste2:=#0;
    end else begin
      if (cx<(x+laenge)) and (taste>=' ') and (taste<#127) then begin
        write (taste);
        Puffer[cx-x+1]:=taste;
        cx:=cx+1;
      end;
      if (taste=#8) and (cx>x) then begin (DEL)
        cx:=cx-1;
        gotoxy (cx,y);
        write (' ');
        puffer[cx-x+1]:=' ';
      end;
    end;
  until (taste=#13) or (taste2=#72) or (taste2=#80);
  textbackground (0);            (bei CPC NORMVIDEO)

  if (format=1) or (format=3) then begin
    i:=laenge;
    while Puffer[i]=' ' do begin
      i:=i-1;
      Puffer:=copy(Puffer,1,i);
    end;
  end;
  if (format=2) or (format=3) then begin
    while (puffer[1]=' ') and (length(Puffer)>0) do begin
      delete(Puffer,1,1);
    end;
  end;
  if gross=1 then for i:= 1 to length(Puffer) do Puffer[i]:=upcase(Puffer[i]);
  if loeschen=1 then begin
    gotoxy (mx,y);
    i:=x-mx+laenge;
    for t := 1 to i do write (' ');
  end;
  gotoxy (altx,alty);
  getstr:=Puffer;
end;


{SYNTAX:        Funktion GETINT (x,y,l,V,A,B,MIN,MAX,L)
 x,y = Position der Eingabebezeichnung (z.B. 'Name:')
 l   = Maximale Laenge der Eingabe (max 80 Zeichen);
 V   = Integervariable enthaelt ursprueglichen Inhalt
 A   = String welcher die Eingabe kennzeichnet (z.B. 'NAME:' oder '')
 B   = Abbruch wenn Cursor nach oben o. unten; CURTASTE enthaelt Tastencode
 MIN = Kleinste moegliche Zahl der Eingabe
 MAX = Groesste moegliche Zahl der Eingabe
 L   = 1 Feld und Feldbeschreibung nach der Eingabe loeschen
       0 Standard Eingabe  }


function getint (x,y,laenge      :byte;
                 Ursprung        :integer;
                 Anzeige         :string80;
                 Abbruch         :byte;
                 Min,Max         :integer;
                 Loeschen        :byte   ):integer;
const
  Reversecolor = LightMagenta;
```

```
Var
  Puffer,Urstring :string80;
  altx,alty,cx,mx:byte;
  i,t,zahl:integer;  (hilfsvariablen)
  taste,taste2:char;
begin
  altx:=wherex;
  alty:=wherey;
  mx:=x;
  str (Ursprung,Urstring);
  Puffer:=copy(Urstring,1,laenge);
  i:=length(Puffer);
  if i<laenge then begin
    i:=laenge-i;
    for t:=1 to i do Puffer:=Puffer+' ';
  end;
  gotoxy (x,y);
  write (Anzeige);
  x:=wherex;
  y:=wherey;

  repeat
    gotoxy (x,y);
    taste2:=#0;
    cx:=x;        (cursor x-position)
    textbackground(Reversecolor);  (bei CPC LOWVIDEO)
    write (Puffer);

    repeat
      if cx=(x+laenge) then begin
        write(^g);
        cx:=cx-1;
      end;
      gotoxy (cx,y);
      read (kbd,taste);
      curtaste:=taste;
      if (taste=#27) and keypressed then begin
        read (kbd,taste);
        curtaste:=taste;
        taste2:=taste;
        if (taste=#75) and (cx>x) then cx:=cx-1;
        if (taste=#77) and (cx<(x+laenge-1)) then cx:=cx+1;
        if (taste=#83)  then begin (DEL unten)
          for i:= 1 to laenge do puffer[i]:=' ';
          cx:=x;
          gotoxy (cx,y);
          write (puffer);
        end;

        if abbruch=0 then taste2:=#0;
      end else begin
        if (cx<(x+laenge)) and (taste in ['0'..'9','-',' ']) then begin
          write (taste);
          Puffer[cx-x+1]:=taste;
          cx:=cx+1;
        end;
        if (taste=#8) and (cx>x) then begin (DEL)
          cx:=cx-1;
          gotoxy (cx,y);
          write (' ');
          puffer[cx-x+1]:=' ';
        end;
      end;
    until (taste=#13) or (taste2=#72) or (taste2=#80);
    textbackground (0);            (bei CPC NORMVIDEO)
    Urstring:=puffer;
    i:=laenge;
    while Puffer[i]=' ' do begin
      i:=i-1;
      Puffer:=copy(Puffer,1,i);
    end;
    while (puffer[1]=' ') and (length(Puffer)>0) do begin
      delete(Puffer,1,1);
    end;
    val (Puffer,Zahl,i);
    Puffer:=Urstring;
  until (i=0) and (zahl>=min) and (zahl<=max);

  if loeschen=1 then begin
    gotoxy (mx,y);
    i:=x-mx+laenge;
    for t := 1 to i do write (' ');
  end;
  gotoxy (altx,alty);
  getint:=zahl;
end;


{SYNTAX:        Funktion GETREAL (x,y,l,k,V,A,B,MIN,MAX,L)
 x,y = Position der Eingabebezeichnung (z.B. 'Name:')
 l   = Maximale Laenge der Eingabe (max 80 Zeichen)
 k   = Anzahl der Nachkommastellen
 V   = REALvariable enthaelt ursprueglichen Inhalt
 A   = String welcher die Eingabe kennzeichnet (z.B. 'NAME:' oder '')
 B   = Abbruch wenn Cursor nach oben o. unten CURTASTE enthaelt Tastencode
 MIN = Kleinste moegliche Zahl der Eingabe
 MAX = Groesste moegliche Zahl der Eingabe
 L   = 1 Feld und Feldbeschreibung nach der Eingabe loeschen
       0 Standard Eingabe  }


function getreal(x,y,laenge,n    :byte;
                 Ursprung        :real;
                 Anzeige         :string80;
                 Abbruch         :byte;
                 Min,Max         :real;
                 Loeschen        :byte   ):real;
const
  Reversecolor = LightMagenta;

Var
  Puffer,Urstring :string80;
  zahl :real;
  altx,alty,cx,mx:byte;
  i,t:integer;  (hilfsvariablen)
  taste,taste2:char;
begin
  altx:=wherex;
  alty:=wherey;
  mx:=x;
  str (Ursprung:laenge-n-1:n,Urstring);
  Puffer:=copy(Urstring,1,laenge);
  i:=length(Puffer);
  if i<laenge then begin
    i:=laenge-i;
    for t:=1 to i do Puffer:=Puffer+' ';
  end;
  gotoxy (x,y);
  write (Anzeige);
  x:=wherex;
  y:=wherey;

  repeat
    gotoxy (x,y);
    taste2:=#0;
    cx:=x;        (cursor x-position)
    textbackground(Reversecolor);  (bei CPC LOWVIDEO)
    write (Puffer);

    repeat
      if cx=(x+laenge) then begin
```

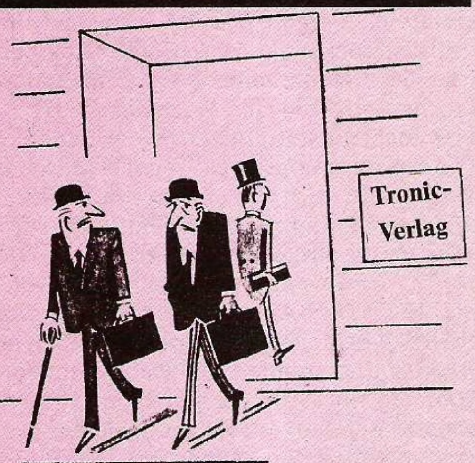
```
        write(^g);
        cx:=cx-1;
      end;
      gotoxy (cx,y);
      read (kbd,taste);
      curtaste:=taste;
      if (taste=#27) and keypressed then begin
        read (kbd,taste);
        curtaste:=taste;
        taste2:=taste;
        if (taste=#75) and (cx>x) then cx:=cx-1;
        if (taste=#77) and (cx<(x+laenge-1)) then cx:=cx+1;
        if (taste=#83)  then begin (DEL unten)
          for i:= 1 to laenge do puffer[i]:=' ';
          cx:=x;
          gotoxy (cx,y);
          write (puffer);
        end;

        if abbruch=0 then taste2:=#0;
      end else begin
        if (cx<(x+laenge)) and (taste in ['0'..'9','-','.',' ']) then begin
          write (taste);
          Puffer[cx-x+1]:=taste;
          cx:=cx+1;
        end;
        if (taste=#8) and (cx>x) then begin (DEL)
          cx:=cx-1;
          gotoxy (cx,y);
          write (' ');
          puffer[cx-x+1]:=' ';
        end;
      end;
    until (taste=#13) or (taste2=#72) or (taste2=#80);
    textbackground (0);            (bei CPC NORMVIDEO)
    Urstring:=puffer;
    i:=laenge;
    while Puffer[i]=' ' do begin
      i:=i-1;
      Puffer:=copy(Puffer,1,i);
    end;
    while (puffer[1]=' ') and (length(Puffer)>0) do begin
      delete(Puffer,1,1);
    end;
    val (Puffer,Zahl,i);
    Puffer:=Urstring;
  until (i=0) and (zahl>=min) and (zahl<=max);

  if loeschen=1 then begin
    gotoxy (mx,y);
    i:=x-mx+laenge;
    for t := 1 to i do write (' ');
  end;
  gotoxy (altx,alty);
  getreal:=zahl;
end;
```

●●● Teil 2 ●●●

```
type demo.pas
program demo;   (Kurze Demonstration der Eingabe-Routinen)

{$i getfeld.inc  Eingabe-Routinen werden eingebunden}

var
  zeitschrift,verlag,ort:string[20];
  postfach,plz:integer;
  heftpreis:real;
  zaehler :integer; (hilfsvariable fuer maske)

begin
  clrscr;
  writeln (' Demonstration der neuen Eingabe-Routinen  (c) F.B. ');
  writeln ('      ---  Computronic  ---              ');

  {Variablen einen bestimmten Wert vorgeben: }

  zeitschrift:='Computronic';
  verlag:='Tronic-Verlag';
  postfach:=870;
  plz:=3440;
  ort:='Eschwege';
  heftpreis:=6,5;

  {Die eigentliche Eingabe: }

  zeitschrift:= getstr(3,4,20,zeitschrift,'Zeitschrift: ',0,3,0,0);
  verlag:= getstr(3,6,20,verlag,'Verlag: ',0,3,0,0);
  postfach:= getint(3,8,5,postfach,'Postfach ',0,0,maxint,0);
  plz:= getint (3,10,4,plz,'PLZ.: ',0,0,9999,0);
  ort:= getstr (15,10,20,ort,'Ort: ',0,3,0,0);
  heftpreis:= getreal (3,12,5,2,heftpreis,'Heftpreis: ',0,1,10,0);

  clrscr;
  writeln ('Das gleiche noch einmal jedoch diesmal mit Feldloeschung:');
  zeitschrift:= getstr(3,4,20,zeitschrift,'Zeitschrift: ',0,3,0,1);
  verlag:= getstr(3,6,20,verlag,'Verlag: ',0,3,0,1);
  postfach:= getint(3,8,5,postfach,'Postfach ',0,0,maxint,1);
  plz:= getint (3,10,4,plz,'PLZ.: ',0,0,9999,1);
  ort:= getstr (15,10,20,ort,'Ort: ',0,3,0,1);
  heftpreis:= getreal (3,12,5,2,heftpreis,'Heftpreis: ',0,1,10,1);

  writeln;
  writeln('Die letzten Eintragungen:');
  writeln;
  writeln (zeitschrift);
  writeln (verlag);
  writeln ('Postfach ',postfach);
  writeln (plz,' ',ort);
  writeln ('Heftpreis: ',heftpreis:2:2);
  writeln;
  writeln ('Bitte Taste druecken!');
  repeat until keypressed;

  clrscr;
  writeln ('Auch richtige Masken sind moeglich:');
  writeln ('Probieren Sie die einmal die Cursortasten RAUF bzw. RUNTER');
  zaehler:=1;
  repeat
    case zaehler of
      1: zeitschrift:= getstr(3,4,20,zeitschrift,'Zeitschrift: ',1,3,0,0);
      2: verlag:= getstr(3,6,20,verlag,'Verlag: ',1,3,0,0);
      3: postfach:= getint(3,8,5,postfach,'Postfach ',1,0,maxint,0);
      4: plz:= getint (3,10,4,plz,'PLZ.: ',1,0,9999,0);
      5: ort:= getstr (15,10,20,ort,'Ort: ',1,3,0,0);
      6: heftpreis:= getreal (3,12,5,2,heftpreis,'Heftpreis: ',1,1,10,0);
    end;
    if curtaste=#72 then zaehler:=zaehler-1
      else zaehler:=zaehler+1;
    if zaehler<1 then zaehler:= 1;
  until zaehler=7;

  clrscr;
  writeln('Die letzten Eintragungen:');
  writeln;
  writeln (zeitschrift);
  writeln (verlag);
  writeln ('Postfach ',postfach);
  writeln (plz,' ',ort);
  writeln ('Heftpreis: ',heftpreis:2:2);
  writeln;
  writeln ('Bitte Taste druecken!');
```

The End

Die Gewinner



Auch diesmal haben die fünf Datenträger Ihre Gewinner gefunden! Dabei kamen wir in der Redaktion uns schon fast vor wie in der Sportschau bei der Auslosung zum Tor des Monats. Ein Berg an Zuschriften hat uns diesmal erreicht. Glücksfee Heike hat folgende fünf Gewinner aus dem Postkarten-Hügel gezogen:

Marlies Kusian, 6588 Dienstweiler
G. Kruse, 2000 Hamburg 74
Michael Symanski, 3000 Hannover 71
Olaf Richter, 1000 Berlin 65
Till Schreiter, 6365 Rosbach 2

Warum gewinnen immer nur die anderen?

Wettbewerb

Spitzenmäßig!

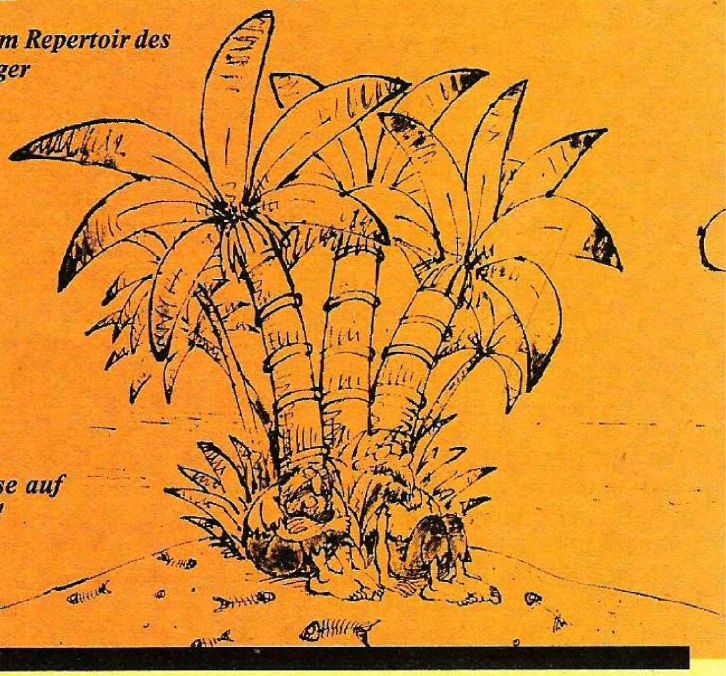
Für nur 60 Pf können Sie wieder eine Programm-Diskette/ Kassette aus dem Repertoir des Tronic-Verlags gewinnen. Insgesamt werden wieder fünf unserer Datenträger verlost. Was Ihr tun müßt, um zu gewinnen? Ganz einfach! Diesmal geht es um urige Sprüche.

So zum Beispiel:
Alle Kinder laufen um das Haus,
nur Klaus
schaut raus!

oder

Alle Kinder spielen auf der Walze,
nur Gunther
liegt drunter!

Auf geht's! Schnappt Euch eine Postkarte und schreibt Eure Sprüche drauf. Vergeßt aber nicht die Bestell-Nr. Eures Programmwunsches mit auf die Karte zu kritzeln, sonst scheidet Ihr schon vorzeitig aus. Das Ganze geht dann an den: Tronic-Verlag; Stad 35; 3440 Eschwege.



Gewinnen Sie eine Reise auf eine einsame Insel, ha! Toller Wettbewerb!

Korrektur

Durch ein kleines Mißgeschick in Form eines Zahlendrehers war der Schneider-Checksummer V 2.0 bislang nicht für den CPC-664 geeignet. Nach folgender kleiner Korrektur ist dieser Fehler nun beseitigt: In Zeile 1270 muß der Wert &96 durch &69 ersetzt werden. Nun können auch die CPC-664-Anwender in den Genuß unseres Checksummers gelangen.

TURBO-PASCAL-KURS Teil 4

Die Flexibilität eines Programms wird besonders durch die im dritten Teil besprochene **IF THEN**-Anweisung gewährleistet: Je nach Bedingung kommt etwas anderes zur Ausführung. Allgemein ausgedrückt könnte man sagen, daß durch solche Kontrollstrukturen die zeilenweise Abarbeitung der Befehle eines Programms unterbrochen wird und eine **Verzweigung** im Programmablauf stattfindet. Bei der FOR TO DO Anweisung geht es weniger um eine Verzweigung als vielmehr um eine **Schleife,** d.h. eine bestimmte Anweisungsfolge wird genauso oft ausgeführt, wie es in der FOR TO DO-Anweisung vorgesehen ist. Sowohl die IF THEN-Anweisung, als auch die FOR TO-Schleife, wird denen, die schon in BASIC programmiert haben, bekannt vorkommen. In PASCAL gibt es noch zwei weitere Befehle, mit denen Schleifen programmiert werden können: die REPEAT-, UNTIL- und die WHILE DO-Strukturen. Darauf werden wir gleich zu sprechen kommen.

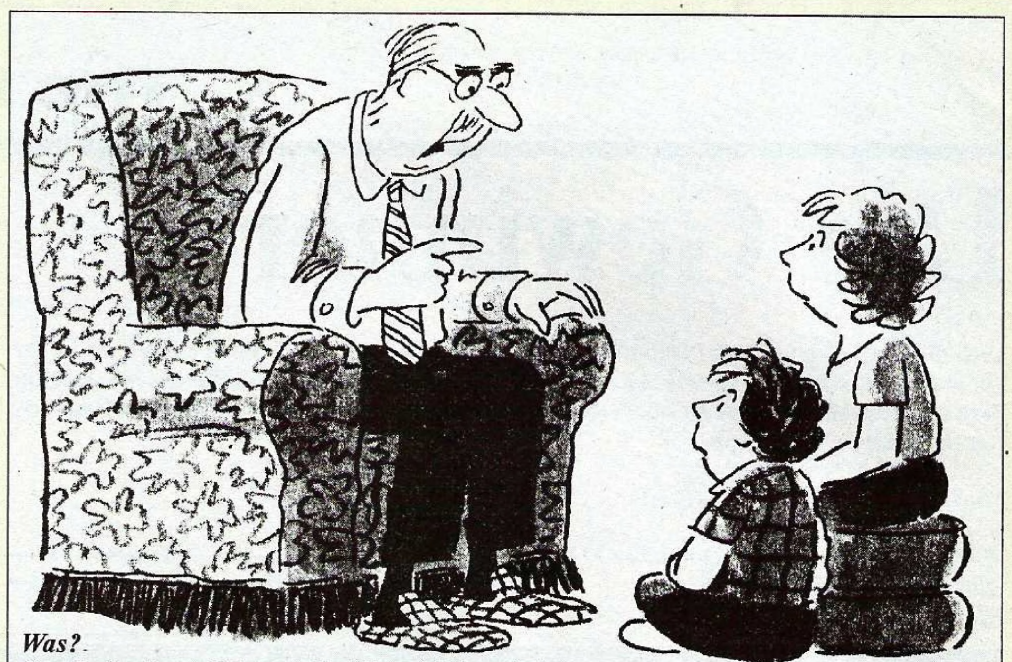
Zunächst wollen wir aber einen neuen Datentyp besprechen, um überhaupt sehen zu können, wozu Kontrollstrukturen wie Schleifen nützlich sein können.

Ein Programm besteht immer aus Datenstrukturen (dem zu bearbeitenden Material) und Algorithmen (den Werkzeugen). Das sehr praktische Werkzeug „Zählschleife" ist für eine spezielle Datenstruktur sehr gut geeignet, die wir noch nicht besprochen haben: das sogenannte **ARRAY** oder auf gut deutsch „Feld" (Wer in Mathematik etwas bewandert ist, kann genauso gut „Vektor" oder „Matrix" sagen).

Ein **ARRAY** ist prinzipiell eine Variable wie jede andere auch – mit einem hochinteressanten Unterschied: sie kann mehr als eine Zahl beinhalten, ja, sie kann unter Umständen tausende von Zahlen beinhalten (um genau zu sein: es muß sich natürlich nicht um Zahlen handeln, es können auch andere Zeichen, kurz, alle Arten von Daten sein). Um die einzelnen Zahlen eines ARRAY noch unterscheiden zu können, bekommt jede Zahl einen Index (deshalb spricht man manchmal auch von indizierten Variablen), d.h. eine Kennziffer, mit der man jede gewünschte Zahl des **ARRAY** wiederfinden kann. Ein ARRAY kann etwa so aussehen:

20 655
11 780
13 735
21 691

In diesem Fall könnte die erste Spalte (also 20, 11, 13, 21) die Startnummer eines Sportlers bei einem Leichtathletikwettkampf bedeuten und die zweite Spalte die Weite, die der/die Sportler/in etwa im Weitsprung gesprungen ist (in cm). Wenn wir diese Werte



Was?
Der Turbo-Pascal-Kurs soll nicht wichtig sein?
In der Zukunft werdet Ihr den alle Nase lang brauchen!

in eine Variable packen wollen, können wir sie z.B. „Weiten" nennen. Der Typ dieser Variablen wäre „ARRAY"; außerdem müssen wir die Größe der Variable bestimmen. Das alles geschieht im Deklarationsteil und würde, wenn genau vier Sportler am Start wären, so aussehen:

weiten: ARRAY[1..4, 1..2] of integer;

Was bedeutet das? Klar ist, daß wir zunächst, um die Variable „weiten" zu deklarieren, angeben müssen, daß es sich um ein ARRAY handelt: das eben tun wir durch die Angabe'ARRAY'. Dann müssen wir angeben, wieviele Zahlen, man nennt sie in diesem Zusammenhang auch Elemente oder **Komponenten,** in das ARRAY passen sollen. Wir haben vier Zeilen und zwei Spalten, also 4 x 2 = 8 Komponenten und schreiben deshalb nach der eckigen Klammer zuerst **1..4**, was soviel heißt wie „die Zeilen dieses ARRAY gehen von 1 bis 4" und anschließend nach einem Komma **1..2** was soviel heißt wie „die Spalten gehen von 1 bis 2". Man sieht also, daß die erste Angabe sich auf Zeilen, die zweite auf Spalten bezieht und die Anzahl der Komponenten eines ARRAY immer die Zeilenzahl multipliziert mit der Spaltenzahl ist.

Um ganz genau zu sein: ein ARRAY muß nicht notwendigerweise aus Zeilen und Spalten bestehen. Wenn es nur aus einer Zeile besteht, kann man die zweite Angabe in der eckigen Klammer weglassen und man bekäme dann eben ein Feld mit einer Zeile bzw. einer Spalte (was keinen Unterschied macht). Außerdem ist man nicht auf Zeilen und Spalten begrenzt: man kann in der eckigen Klammer noch mehr Angaben machen – eine dritte Angabe der Form 1..10 könnte man z.B. als weitere zehn Spalten oder „Säulen" interpretieren, die zu den vorherigen Zeilen und Spalten senkrecht stehen und prinzipiell könnte man auf diese Weise ARRAY mit vielen Dimensionen herstellen. Den mathematisch nicht so versierten unter euch wird das etwas spanisch vorkommen, wir erwähnen es nur der Vollständigkeit halber und werden vorläufig nur mit ARRAYs arbeiten, die eine bestimmte Anzahl von Spalten und Zeilen haben.

Erinnern wir uns an den Vergleich einer Variable mit einer Schublade: die Schublade hat einen Namen und einen Inhalt, den Wert. Wir sind bisher immer davon ausgegangen, daß jede Schublade (Variable) genau einen Wert beinhalten kann. Der ARRAY-Typ zeigt, daß dem nicht so sein muß. Wir können uns ein ARRAY als eine Schublade mit vielen Fächern vorstellen, jedes Fach der Schublade hat eine Kennziffer, den **Index,** und einen entsprechenden Inhalt, eben ihren Wert. Wichtig ist, daß bei ARRAYs alle Komponenten (d.h. alle Fächer einer ARRAY-Schublade) vom gleichen Typ sein müssen: ich kann also nicht in ein Fach eine integer-Zahl hineinlegen und in ein anderes Fach ein char-Zeichen. Wenn wir einer Komponente eines ARRAY einen Wert zuweisen wollen, müssen wir sie über ihren **Index** ansprechen, z.B.

weiten [1,2]:=655;

d.h. der Komponente in der ersten Zeile **[1,2]** und zweiten Spalte [1,2] wird der Wert 655 zugewiesen. Genauso können wir natürlich auch abfragen, was in einer be-

stimmten Zeile und Spalte steht, z.B. mit writeln (weiten [3,1]; (das Ergebnis wäre 13).
Vielleicht ist euch inzwischen auch schon klar geworden, warum eine Zählschleife zur Bearbeitung von ARRAYs besonders nützlich ist. Haben wir zum Beispiel eine Variable so definiert:
noten: ARRAY[1..6] of integer;
dann bedeutet das, daß die Variable noten 6 Zeilen (und nur eine Spalte hat), also 6 Komponenten aufnehmen kann. Wenn wir wissen wollen, welcher Wert in jeder Komponente steht, wäre es Unsinn, jedes Element einzeln anzusprechen, einfacher geht es so
for i:=1 to 6 do
writeln(noten[i];
Was passiert? Die Schleife zählt von 1 bis 6. Nach jedem Zählschritt hat die Zählvariable i den entsprechenden Wert (nämlich zuerst 1, dann 2, dann 3...). Dieser Wert von i wird dazu benutzt, ihn als Index für die Variable „noten" einzusetzen: notenÄiÜ - im ersten Zählschritt wäre das äquivalent mit noten[1] und wir bekämen den Wert der Komponente in der ersten Zeile von „noten"ausgedruckt. Und so ginge es weiter bis i den Wert 6 hat. Dann wird die Schleife verlassen und der nächste Programmbefehl ausgeführt.
Wir wollen ein kleines Beispielprogramm schreiben. Nehmen wir als Ausgangspunkt das Programm „quadrat-und-wurzel" aus Teil 3. Das jetzige Programm soll ebenfalls für alle Zahlen von 1-10 das Quadrat und die Wurzel berechnen, die Werte aber nicht ausdrucken, sondern in einem Feld abspeichern. In diesem Feld soll in der ersten Spalte die Zahl, in der zweiten ihr Quadrat und in der dritten Spalte die Wurzel der Zahl stehen.

```
 1: PRGRAM quadratwurzelfeld;
 2: VAR i, j: integer;
 3: feld: ARRAY [1..10,1..3] of real;
 4: BEGIN
 5: clrscr;
 6: FOR i : = 1 TO 10 DO
 7: BEGIN
 8: feld [i,1] : = i;
 9: feld [i,2] : = sqr(i);
10: feld [i,3] : = sqrt(i)
11: END
12: END
```

In Zeile 7 bis 11 steht eine Verbundanweisung, die zehnmal durchlaufen wird: bei jedem Durchlauf ist die Variable i (die Zählvariable) um 1 erhöht und entsprechend wird das ARRAY 'feld' mit i und den Werten des Quadrats und der Wurzel von i gefüllt. Die Definition der Variable 'feld' in Zeile 3 beschreibt 'feld' als eine Schublade mit 10 Zeilen und 3 Spalten und der Typ von Zahlen, der in den Fächern abgelegt werden kann, sind real-Zahlen.
Will man sich die Werte noch ausdrucken lassen, kann man Zeile 11 folgenden Anhang mit ins Programm aufnehmen:

```
12: FOR i: = 1 TO 10 DO
13: FOR j: = 1 TO 3 DO
14: writeln (feld [i, j];
```

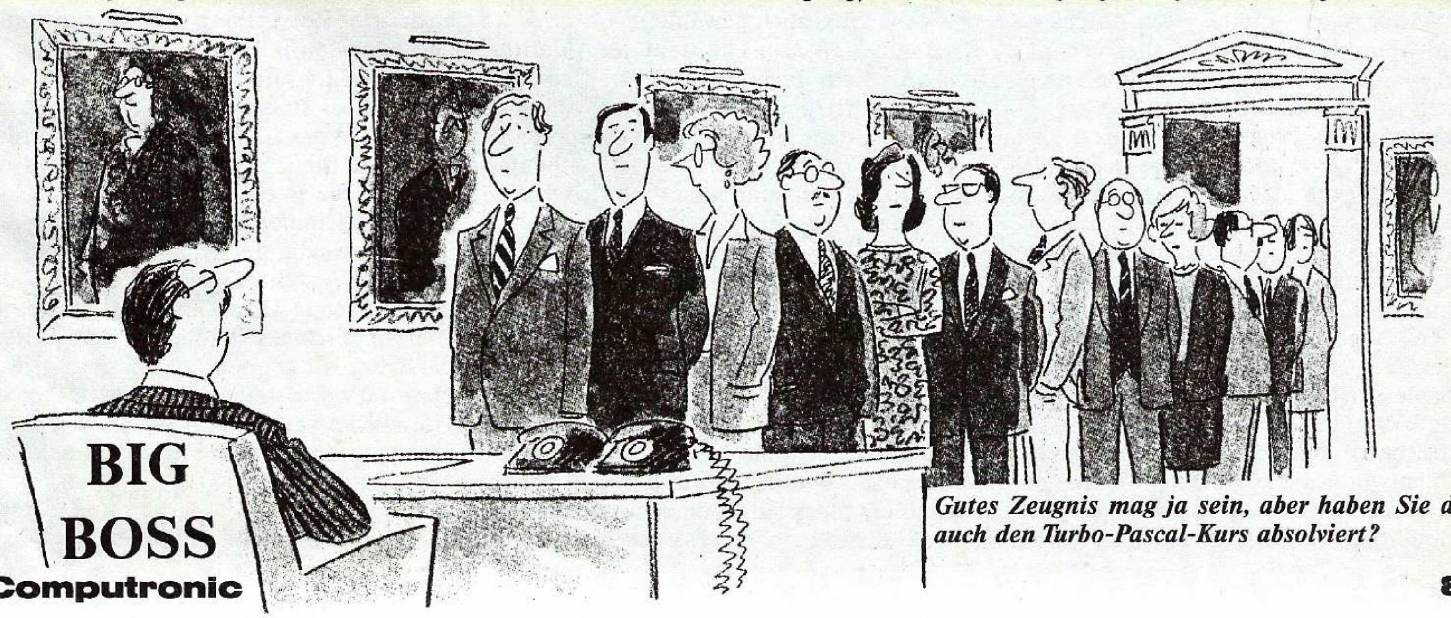
Statt einer FOR TO DO Schleife ist es oft hilfreicher, eine REPEAT UNTIL Anweisung zu verwenden. REPEAT UNTIL heißt auf gut deutsch soviel wie: Wiederhole dies und das solange, bis dies oder jenes der Fall ist.
z.B.:
Wiederhole folgendes:
„Schreibe i
Erhöhe i um 1„
bis i den Wert 10 hat.
Wir hätten damit eine FOR TO Schleife nachgebaut. Der entscheidene Vorteil von REPEAT UNTIL ist, daß die Anzahl der Schleifendurchläufe nicht von vornherein feststehen muß, sondern nach jedem Durchlauf wird eine Bedingung, die hinter dem Wort UNTIL steht, geprüft und je nachdem die Schleife nochmals durchlaufen. Sehen wir uns folgendes Programmstück an:

```
1: REPEAT
2: write ('Gib eine Zahl ein ');
3: readln (zahl);
4: writeln ('Das Quadrat davon ist ',
   sqr(zahl))
5: UNTIL zahl = 0;
```

Was passiert? Eine Zahl wird eingegeben (Zeile 3) und das Quadrat der Zahl wird ausgegeben (Zeile 4). Dann, in Zeile 5, wird geprüft, ob die ZAHL DEN Wert 0 hat: wenn ja, wird die Schleife abgebrochen, wenn nicht, wird die Schleife ab Zeile 2 nochmals durchlaufen und so weiter bis der Wert 0 eingegeben wurde.
Halten wir folgende Eigenschaften von REPEAT UNTIL fest: Es ist eine Schleife, die mindestens einmal durchlaufen wird. Am Ende der Schleife, bei UNTIL, wird geprüft, ob die Schleife nochmals durchlaufen werden soll. Zwischen REPEAT und UNTIL können beliebig viele Anweisungen stehen, die (und das ist für PASCAL etwas seltsam) nicht(!) als Verbundanweisung stehen müssen (also nicht mit BEGIN und END eingeschlossen sein müssen).
Ähnlich und doch auch anders ist die WHILE DO Schleife. Ihr Verhalten kann etwa so beschrieben werden:
Solange dies und das wahr ist, mache das folgende: „bla bla...„
In Pascalianisch:

```
1: WHILE i 10 DO
2: BEGIN
3: write (i);
4: writeln ('Das Quadrat: ',sqr(i));
5: END;
```

Die Unterschiede zu REPEAT UNTIL liegen auf der Hand: Die Prüfung einer Bedingung erfolgt am Anfang und nicht am



Gutes Zeugnis mag ja sein, aber haben Sie denn auch den Turbo-Pascal-Kurs absolviert?

Ende der Schleife. Das hat zur Folge, daß, im Gegensatz zu REPEAT UNTIL, die Schleife nicht unbedingt einmal durchlaufen werden muß. Stößt das Programm im Ablauf auf eine WHILE Anweisung und die Prüfung der Bedingung ergibt, daß sie falsch ist, so wird die ganze WHILE DO Schleife übersprungen. Der zweite Unterschied ist auch sofort zu sehen: Die Anweisungen, die innerhalb der Schleife ausgeführt werden sollen, müssen, wenn es mehr als eine Anweisung ist, mit BEGIN und END als Verbundanweisung eingeschlossen werden.

Für die REPEAT Anweisung soll noch ein kleines, aber sehr nützliches Beispiel folgen. Manchmal will man den Programmablauf unterbrechen, z.B. um sich ein Ergebnis in Ruhe anzusehen, und durch Drücken einer beliebigen Taste soll es weitergehen. Das erreicht man durch folgende Programmzeile:
REPEAT UNTIL KEYPRESSED;

Zwischen REPEAT und UNTIL steht „nichts", also macht das Programm auch nichts, d.h. es bleibt einfach stehen. Und zwar solange, bis die Funktion KEYPRESSED wahr ist. KEYPRESSED ist eine eingebaute PASCAL Funktion, die überprüft, ob eine Taste gedrückt wurde. Wenn dies der Fall war, ist KEYPRESSED wahr (hat den Wert TRUE) und folglich wird die REPEAT Schleife beendet.

Wir wollen unsere neuen Kenntnisse in einem etwas längeren Programm zusammenfassen.

```
 1:   PROGRAM x_hoch_y;
 2:   VAR x,resultat: real;
 3:       y,zaehler,i: integer;
 4:       speicher: ARRAY[1..20,1..3] of real;
 5:       antwort: char;
 6:   BEGIN
 7:        zaehler:=0; antwort:='j';
 8:        REPEAT
 9:             clrscr;
10:             zaehler:=zaehler+1;
11:             writeln('Dieses Programm berechnet das Ergebnis
                         fuer die');
12:             writeln('mathematische Funktion x hoch y fuer den
                         Fall,');
13:             writeln('dass x und y ganze Zahlen sind');
14:             writeln;
15:             writeln('Die Ergebnisse werden in einem
                         Array gespeichert.');
16:             writeln;
17:             write('Eingabe von x: '); readln(x);
18:             write('Eingabe von y: '); readln(y);
19:             writeln('Es wird berechnet: ',x:0:2,' hoch ',y);
20:             writeln;
21:             IF y=0 THEN resultat:=1;
22:             IF y=1 THEN resultat:=x
23:             ELSE BEGIN
24:                       i:=1;
25:                       resultat:=1;
26:                       WHILE i<=y DO
27:                            BEGIN
28:                                 resultat:=resultat*x;
29:                                 i:=i+1
30:                            END
31:                  END;
32:             writeln('Das Ergebnis lautet: ',resultat:10:3);
33:             writeln;
34:             speicher[zaehler,1]:=x;
35:             speicher[zaehler,2]:=y;
36:             speicher[zaehler,3]:=resultat;
37:             writeln('Wollen Sie noch eine Rechnung durchfuehren?
                         (j/n)');
38:             readln(antwort)
39:        UNTIL (antwort='n') or (zaehler=20);
40:        write('Feld ausdrucken ? (j/n): '); readln(antwort);
41:        clrscr;
42:        IF antwort='j'
43:        THEN FOR i:=1 TO zaehler DO
44:                 writeln(speicher[i,1]:0:2,' hoch ',
                         speicher[i,2]:0:0,' = ',speicher[i,3]:0:2);
45:   END.
```

Tippt das Programm am besten ab, compiliert es und laßt es ein paar mal laufen, um zu sehen, was passiert. Dies ist schon ein ziemlich langes Programm und wir werden die wichtigsten Punkte jetzt durchsprechen. Wenn ihr nicht sofort alles versteht, ist das nicht schlimm. Programme zu verstehen ist auch eine Übungssache.
Was macht das Programm? Es liest zwei Zahlen ein, z.B. 2 und 4 und berechnet das Ergebnis für die mathematische Funktion 2 (hoch 4) (wer es nicht mehr weiß: 2 hoch 4 bedeutet, daß die 2 viermal mit sich selbst malgenommen wird, also 2x2x2x2, das Ergebnis ist 16). Das Ergebnis wird in einem ARRAY abgespeichert.
In den Zeilen 2 bis 5 werden die benötigten Variablen definiert (Deklarationsteil). In Zeile 6 beginnt das Hauptprogramm. Zunächst bekommen die Variablen „zaehler" und „antwort" einen Wert zugewiesen (Zeile 7). In Zeile 8 beginnt die REPEAT-Schleife. Sie endet erst (Zeile 39), wenn der Benutzer auf die Frage „Wollen Sie noch eine Rechnung durchführen?" mit „n" antwortet oder, wenn schon 20 Berechnungen durchgeführt wurden. (Übrigens ist die Zahl 20 willkürlich gewählt!)
In den Zeilen 21 bis 31 steht der Algorithmus für die Berechnung. Hier werden drei Fälle unterschieden:
1: y ist 0, dann ist das Ergebnis immer 1, egal welchen Wert x hat (das ist mathematisch so definiert und man muß sich damit abfinden)(Zeile 21)
2: y ist 1, dann ist das Ergebnis immer der Wert von x, egal welcher Wert das ist (z.B. 545 (hoch 1) = 545) (Zeile 23)
3: y ist größer als 1, dann wird x eben y-mal mit sich selbst malgenommen (Zeile 23 bis 31)
Versucht am besten, das Programm mit einem einfachen Beispiel (2 hoch 4) schrittweise nachzuvollziehen und notiert dabei auf einem Stück Papier die entsprechenden Werte, die die einzelnen Variablen gerade bekommen. Dadurch bekommt man ein Gefühl für die Logik des Algorithmus. (Dies nennt man übrigens den sog. „Schreibtischtest", der sehr wichtig ist, um auch die Korrektheit von Algorithmen nachzuprüfen.)
In Zeile 32 schließlich wird das Ergebnis ausgegeben. In Zeile 34 bis 36 passiert folgendes: In das ARRAY „speicher" werden die Werte für „x", „y" und „resultat" so eingelesen, daß in einer **Zeile** in der **1. Spalte** der x-Wert, in der **2. Spalte** der y-Wert, und in der **3. Spalte** der Wert des Ergebnisses („resultat") steht. Dies wird benutzt, um anschließend (Zeile 43 und 44) die gespeicherten Werte als Tabelle auf dem Schirm darstellen zu können.
Das war's für heute! Versucht möglichst viel mit Pascal auf dem Rechner zu üben. Übung macht auch im Programmieren den Meister.

Ende Teil 4

TURBO-PASCAL-KURS Teil 5

Datentypen

Heute wollen wir uns genauer mit den Datentypen in Turbo-Pascal beschäftigen. Die Standard-Datentypen haben wir schon einmal aufgezählt (siehe Folge 2), wollen sie aber kurz wiederholen:

INTEGER:
Ganze Zahlen im Bereich von -32768 bis 32767.

BYTE:
Ganze Zahlen im Bereich von 0 bis 255. (Byte umfaßt also einen Teil des INTEGER-Bereichs)

REAL:
Dezimalzahlen im Bereich von 1E-38 bis 1E38 (wenn Dir diese Zahlen unbekannt vorkommen, dann schau mal in die Folge 2 rein).

CHAR:
alphanumerische Zeichen, also das Alphabet (Groß- und Kleinbuchstaben), Ziffern (beachte, daß man mit diesen Ziffern nicht rechnen kann, sie gelten nicht als Zahlen!) und Sonderzeichen (z.B. „+", „:" etc.)

BOOLEAN:
Die sogenannten Wahrheitswerte TRUE (wahr) oder FALSE (falsch).
Über diese Standard-Typen hinaus bietet Pascal die Möglichkeit an, selbst Datentypen zu definieren. Der Ort, wo das geschieht, ist im Deklarationsteil. Die Regel dafür lautet:

TYPE Name = Datentyp

TYPE ist ein reserviertes Wort, mit dem die Definition von Datentypen eingeleitet wird. Nach TYPE können dann mehrere Definitionen von Datentypen erfolgen. Jede einzelne Definition wird mit einem Semikolon (;) abgeschlossen.

Der Aufzählungstyp

Bsp.: TYPE Spielfarbe = (karo,herz,pik, kreuz);
Hier wird ein Datentyp definiert, der aus 4 Werten besteht (1.karo, 2. herz, 3. pik, 4. kreuz). Dieser spezielle Datentyp bekommt den Namen „Spielfarbe". Jetzt kann eine Variable „Trumpf" folgendermaßen definiert werden:

VAR Trumpf: Spielfarbe;

Die Variable „Trumpf" kann jetzt alle Werte annehmen, die im Datentyp „Spielfarbe" enthalten sind (also: karo, herz, pik und kreuz).
Diese Art der Definition eines Datentyps nennt man deshalb Aufzählungstyp, weil einfach in Klammern alle Werte aufgezählt werden, die im Datentyp enthalten sind.
Die Werte des Datentyps haben durch ihre Aufzählung auch eine Ordnung zugewiesen bekommen. Das bedeutet, daß der erste Wert der kleinste ist und der letzte ist der größte:

karo < herz; herz < pik; kreuz > pik;

Folgende Vergleiche können auch angestellt werden:

karo <> kreuz (ungleich)
herz = herz (gleich)
karo <= herz (kleiner oder gleich)
pik >= herz (größer oder gleich)

In Pascal bedeutet das, daß jedem der vier Werte eine **Ordnungszahl** zugewiesen wird (das macht Pascal ganz alleine). Die Zählung beginnt immer beim ersten Wert mit 0. Es gilt also: karo hat die Ordnungszahl 0, herz hat 1, pik hat 2, kreuz hat 3. Du kannst auch Turbo-Pascal mit Hilfe eines kleinen Programms befragen, ob folgendes gültig ist:

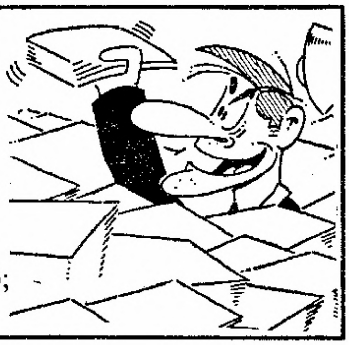
so beschreiben: Jeder Wert, mit Ausnahme des Letzten, hat einen ***Nachfolger*** (z.B. ist „pik" der Nachfolger von „herz") und jeder Wert, mit Ausnahme des ersten, hat einen ***Vorgänger*** (z.B. ist „pik" der Vorgänger von „kreuz"). Für den ***Nachfolger*** bzw. den ***Vorgänger*** gibt es in Pascal zwei Funktionen. Mit ***pred(Variable)*** (vom Englischen „predecessor" = Vorgänger) erhält man den Vorgänger der Variablen, mit ***succ(Variable)*** (vom Englischen „successor" = Nachfolger) erhält man den Nachfolger der Variablen.

Bsp.:

trumpf := pred(herz) („trumpf" enthält danach den Wert „karo")
trumpf := succ(herz) („trumpf" enthält jetzt den Wert „pik")

pred(karo) und succ(kreuz) sind nicht definiert
Mit einigen Beispielprogrammen soll nun demonstriert werden, was man mit solchen selbstgebastelten Datentypen machen kann. Aber zunächst ein kleiner Wermutstropfen. Etwas kann man nämlich nicht machen. Der Inhalt unserer Variablen „Trumpf" mit dem Datentyp „Spielfarbe" (so wie oben definiert), kann nicht ausge-

```
1:  PROGRAM Ordnungszahl_zeigen;
2:  TYPE Spielfarbe = (karo,herz,pik,kreuz);
3:  VAR Trumpf : Spielfarbe;
4:  BEGIN
5:      clrscr;
6:      writeln('Ordnungszahl von „karo" : ',ord(karo));
7:      writeln('Ordnungszahl von „herz" : ',ord(herz));
8:      writeln('Ordnungszahl von „pik"  : ',ord(pik));
9:      writeln('Ordnungszahl von „kreuz": ',ord(kreuz));
10: END.
```



Viel Neues steckt nicht im Programm drin. Bestimmt hast Du auch schon erkannt, daß die Standard-Funktion, mit der man den Ordnungswert erfahren kann, „ord(Wert)" lautet. Ein Hinweis noch: Im Deklarationsteil darf die Zeile 2 nicht mit der Zeile 3 vertauscht werden. Dies würde einen Fehler beim Compilieren ergeben, weil der Datentyp Spielfarbe dann noch gar nicht bekannt wäre.
Der allgemeine Merksatz dafür ist bei Pascal: Alles, was benutzt wird, muß zum Zeitpunkt des Auftretens definiert worden sein. Das ist insofern auch einsichtig, weil der Compiler beim Compilieren nicht „nach vorne schauen" kann.

Die Ordnung der vier Werte in unserem selbstdefinierten Datentyp kann man auch

druckt werden. Bei folgender Anweisung gibt es einen Fehler beim Compilieren:
Trumpf := karo;
writeln (Trumpf);
Um den Inhalt solch einer Variablen auszudrucken, muß man immer einen Umweg beschreiten (z.B. mit der CASE-Anweisung, wie in unseren Beispielen).

Alle Beispielprogramme machen das gleiche, nämlich die Auflistung aller möglichen Werte von „Trumpf" (einmal vom kleinsten zum größten Wert, einmal umgekehrt), jedoch auf verschiedene Art und Weise. Gleichzeitig ist dies eine Wiederholung für die ***Schleifen*** in Pascal. In allen Programmen wird der jeweils nächste Wert immer erst ausgegeben, wenn eine beliebige Taste gedrückt wurde.

TURBO-PASCAL ★ TURBO-PASCAL

```
 1:  PROGRM Datentyp_Spielfarbe_1;
 2:  TYPE Spielfarbe = (karo,herz,pik,kreuz);
 3:  VAR Trumpf: spielfarbe;
 4:  BEGIN
 5:      clrscr;
 6:      FOR trumpf:= karo TO kreuz DO
 7:      BEGIN
 8:          CASE Trumpf OF
 9:           karo: writeln('karo');
10:           herz: writeln('herz');
11:           pik: writeln('pik');
12:           kreuz: writeln('kreuz');
13:         END;
14:         REPEAT UNTIL keypressed { Das Programm wartet, bis
15:                                   eine Taste gedrueckt wurde }
16:     END
17:  END.
```

Hier wird eine WHILE-Schleife eingesetzt, um die Auflistung der Werte zu erzielen. Interessant ist hier die Zeile 6. Hier wird der Variablen „Trumpf" der Nachfolger von „kreuz" zugewiesen, den es ja gar nicht gibt. So erstaunlich es sein mag – das funktioniert, denn dieser Nachfolger von „kreuz" soll ja nicht ausgedruckt werden. Dadurch wird aber gewährleistet, daß durch die Verwendung von „pred(Trumpf)" in Zeile 9 wirklich alle Werte ausgedruckt werden können (denn: der Vorgänger vom Nachfolger von kreuz ist genau wieder kreuz!! Ruhig mehrmals durchlesen). Bei solchen Tricks muß man sich jedoch immer genau überlegen, ob nicht irgendwo im Programm ein Fehler entstehen kann. Kennenlernen sollte man solche Möglichkeiten aber durchaus.

Du siehst, daß man die FOR-Anweisung genau so benutzen kann, wie z.B. in Folge 3 mit INTEGER-Zahlen (Bsp.: FOR i:= 1 to 5 DO). Das Programm müßte eigentlich verständlich sein. Neu ist der Text in Zeile 14, der in geschweiften Klammern steht. Dies ist ein ***Kommentar,*** der für die Programmausführung keine Bedeutung hat. Statt einer geschweiften Klammer kann man auch für den Kommentar anfangs „(*" (runde Klammer mit Stern) und für das Kommentarende „*)" schreiben. Kommentare sind übrigens sehr wichtig für die Dokumentation eines Programms und machen ein Programm verständlich, auch wenn man es selbst nicht geschrieben hat oder aber auch, wenn es schon längere Zeit her ist, daß man es geschrieben hat. Die geniale Lösung eines Problems, die man beim Schreiben eines Programms gefunden hatte, versteht man vielleicht selbst nicht mehr, wenn man sich das Programm nach einem Jahr anschaut, um etwas daran zu ändern.

```
 1:  PROGRAM Datentyp_Spielfarbe_3;
 2:  TYPE Spielfarbe = (karo,herz,pik,kreuz);
 3:  VAR Trumpf: spielfarbe;
 4:  BEGIN
 5:      clrscr;
 6:      Trumpf:= pred(karo);
 7:      WHILE Trumpf <> kreuz DO
 8:      BEGIN
 9:          Trumpf:= succ(Trumpf);
10:          CASE Trumpf OF
11:           karo: writeln('karo');
12:           herz: writeln('herz');
13:           pik: writeln('pik');
14:           kreuz: writeln('kreuz');
15:         END;
16:         REPEAT UNTIL keypressed  { Das Programm wartet, bis
17:                                    eine Taste gedrueckt wurde }
18:     END
19:  END.
```

Dieses Programm arbeitet genau umgekehrt zum Programm 2.

```
 1:  PROGRAM Datentyp_Spielfarbe_2;
 2:  TYPE Spielfarbe = (karo,herz,pik,kreuz);
 3:  VAR Trumpf: spielfarbe;
 4:  BEGIN
 5:      clrscr;
 6:      Trumpf:= succ(kreuz);
 7:      WHILE Trumpf <> karo DO
 8:      BEGIN
 9:          Trumpf:= pred(Trumpf);
10:          CASE Trumpf OF
11:           karo: writeln('karo');
12:           herz: writeln('herz');
13:           pik: writeln('pik');
14:           kreuz: writeln('kreuz');
15:         END;
16:         REPEAT UNTIL keypressed { Das Programm wartet, bis
17:                                   eine Taste gedrueckt wurde }
18:     END
19:  END.
```

Toll, dieser Turbo-Pascal-Kurs!

```
 1:  PROGRAM Datentyp_Spielfarbe_4;
 2:  TYPE Spielfarbe = (karo,herz,pik,kreuz);
 3:  VAR Trumpf: spielfarbe;
 4:  BEGIN
 5:      clrscr;
 6:      Trumpf:= pred(karo);
 7:      REPEAT
 8:         Trumpf:= succ(Trumpf);
 9:         CASE Trumpf OF
10:           karo: writeln('karo');
11:           herz: writeln('herz');
12:           pik: writeln('pik');
13:           kreuz: writeln('kreuz');
14:         END;
15:         REPEAT UNTIL keypressed { Das Programm wartet, bis
16:                                   eine Taste gedrueckt wurde }
17:     UNTIL Trumpf = kreuz
18:  END.
```

Hier wird gezeigt, daß auch die REPEAT-Schleife mit solchen Datentypen benutzt werden kann. Ansonsten ist alles gleich, wie bei Programm 3.

```
 1:  PROGRAM Datentyp_Spielfarbe_5;
 2:  TYPE Spielfarbe = (karo,herz,pik,kreuz);
 3:  VAR Trumpf: spielfarbe;
 4:  BEGIN
 5:      clrscr;
 6:      Trumpf:= succ(kreuz);
 7:      REPEAT
 8:         Trumpf:= pred(Trumpf);
 9:         CASE Trumpf OF
10:           karo: writeln('karo');
11:           herz: writeln('herz');
12:           pik: writeln('pik');
13:           kreuz: writeln('kreuz');
14:         END;
15:         REPEAT UNTIL keypressed { Das Programm wartet, bis
16:                                   eine Taste gedrueckt wurde }
17:     UNTIL Trumpf = karo
18:  END.
```

Die Umkehrung zu Programm 4.

Sinn der Typenvereinbarung

Vielleicht hast Du Dich gefragt, wozu das eigentlich gut sein soll, diese Typenvereinbarung, und ob das nicht einfacher geht. Dazu ganz kurz ein paar Anmerkungen. Wir könnten uns die Typenvereinbarung unserer Spielfarbe durchaus sparen und das ganze Problem anders lösen. Wir könnten davon ausgehen, daß „karo" z.B. den Wert 1 bekommt, „herz" den Wert 2 usw :a Man hätte nur die Variable „trumpf", die den Datentyp INTEGER hätte. Im Programm könnte dann z.B. stehen:

```
Bsp. 1   IF trumpf = 2
         THEN writeln('Herz');
```

Diese Anweisung hat mit Sicherheit den gleichen Effekt wie diese:

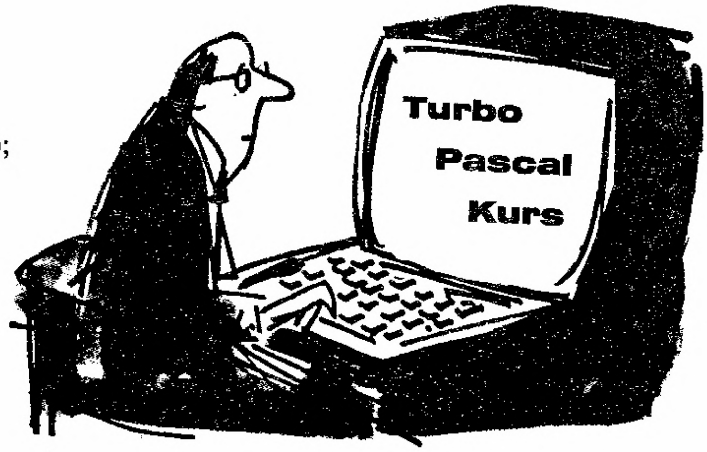
```
Bsp. 2   IF trumpf = herz
         THEN writeln('Herz');
```

Der Vorteil des selbstdefinierten Datentyps ist, daß ein Programm viel besser lesbar wird und damit auch leichter kontrollierbar und änderbar ist. Im Beispiel 2 sieht man eben sofort, daß die Anweisung richtig ist, während man beim Beispiel 1 noch zusätzlich wissen muß, welchen Zahlenwert „herz" hat. Nun stell Dir mal ein umfangreiches Programm vor, in dem mehrere solche Fälle vorkommen, und es wimmelt nur von Zahlen, die für irgendwelche Begriffe stehen. Da kann man dann leicht den Überblick verlieren.

Der Ausschnittstyp

Ein ***Ausschnittstyp*** ist ein Teilbereich eines bereits definierten Datentyps. Ein Beispiel kennt Ihr schon: BYTE ist nämlich solch ein Ausschnittstyp.

Zu beachten ist, daß der Ausschnittstyp sich immer auf einen schon vorhandenen Datentyp bezieht (der kann selbst definiert sein oder einer der Standard-Datentypen von Turbo-Pascal sein – jedoch nicht der Datentyp REAL). Der Ausschnittstyp darf auch nicht größer sein als der schon vorhandene Datentyp (das ist ja logisch!).



All' meine Kollegen beherrschen Turbo-Pascal, nur ich nicht! Hoffentlich finde ich wieder Anschluß!

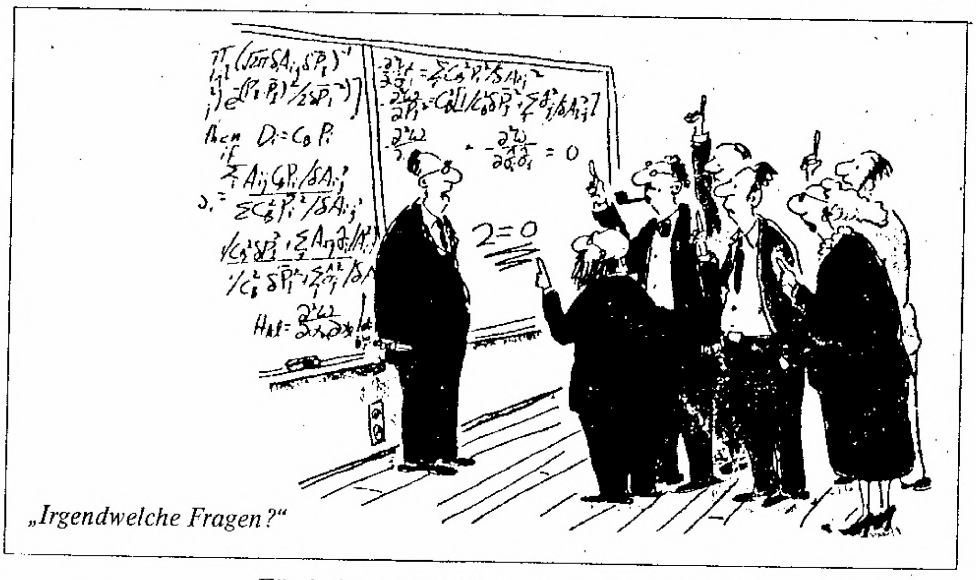
Die Regel lautet:
TYPE Name =
untere_Grenze .. obere_Grenze;
Einige Beispiele (der Grundtyp ist jeweils ein Standard-Typ):

```
TYPE ziffern = '0'..'9'; (Grundtyp: char)
zahlen   = 1 .. 100;  (Grundtyp: integer)
noten    = 1 ..   6;  (Grundtyp: integer)
grossbuchstaben =*
*'A' .. 'Z'  (Grundtyp: char)
```

Beim Arbeiten mit Variablen eines Ausschnittstyps muß darauf geachtet werden, daß der definierte Bereich nicht überschritten wird.

Folgende Anweisung ist zwar zulässig, kann jedoch zu einem Fehler während der Programmausfahrung (Laufzeitfehler) führen:

```
readln(eingabe); (* „eingabe" ist vom Da-
                    tentyp „zahlen" *)
eingabe:=        eingabe + 10;
```



„Irgendwelche Fragen?"

Für jede „eingabe", die größer als 90 ist, führt die Zuweisung „eingabe:=eingabe+10" zu einem Laufzeitfehler, weil der definierte Bereich überschritten wird.

Hier ist man also wieder einmal gezwungen, sich über den Programmablauf einige Gedanken zu machen, was jedoch dem Programm bestimmt nicht schadet. Der Vorteil dieser Ausschnitttypen ist wieder die zu erzielende Programmklarheit. Dazu ein Beispiel:

```
TYPE wochentag= (mon,die,mit,don,fre,sam,son);
     werktag      = mon .. fre;
VAR  arbeitstag  : werktag;
```

Ich muß Euch warnen, ich kann Turbo-Pascal!

Hier spricht die Variable wirklich für sich und bedarf keiner weiteren Erklärung. Auch ist gesichert, daß eine falsche Zuweisung wirklich zu einem Fehler führt und somit vom Programmierer nicht unter Umständen „übersehen" werden kann.

Der Datentyp STRING

Turbo-Pascal bietet den Datentyp ***STRING*** an, um Zeichenketten zu verarbeiten. Der Datentyp STRING hat etwas Ähnlichkeit mit dem ARRAY (s. Folge 4).

Die Regel lautet:

```
TYPE Name = STRING [Num];
```

Dranbleiben!

Auf zum Turbo-Pascal Kws

STRING ist ein reserviertes Wort. „Num" ist ein INTEGER-Wert zwischen 1 und 255 und gibt die Zahl der Zeichen an, aus denen der String maximal bestehen kann.

Beispiele:

```
TYPE Zeile  = STRING[80];
     Name   = STRING[20];
VAR  Nachname, Vorname: Name;
     Bemerkung        : Zeile;
```

Je länger der STRING definiert wird, desto mehr Speicherplatz wird beim Compilieren reserviert.

Der Datentyp ARRAY

Was ein ARRAY ist, haben wir ja schon besprochen. Hier geht es nur darum zu zeigen, daß man auch einen Datentyp als ARRAY definieren kann.

„...Ich komme damit zum Ende meiner kurzen Ausführungen..."

Regel: TYPE Name = ARRAY [Anfang .. Ende] of Datentyp;

Beispiele:

```
CONST  Maximum = 100;
TYPE   Zahlenfeld = ARRAY [1 .. Maximum] of INTEGER;
       Wochentag = (mon,die,mit,don,fre,sam,son);
       Ausgaben = ARRAY [Wochentag] of real;
VAR    Wert : Zahlenfeld;
       Ausg : Ausgaben;
```

Wie man auf die einzelnen Komponenten eines Feldes zugreift, wurde auch schon besprochen. Möglich ist bei der Variablen „Wert" folgendes:

```
Wert[1] := 5;
Wert[2] := 8;
```

Vielleicht hast Du Probleme, Dir die Variable „Ausg" vorzustellen. Stell Dir 7 Kästchen nebeneinander vor. In jedes Kästchen kann ein Wert vom Typ REAL gelegt werden. Die Kästchen haben die Namen „mon" bis „son".

mon	die	mit	don	fre	sam	son

Auf die einzelnen Komponenten (Kästchen) dieses Feldes kann folgendermaßen zugegriffen werden:

```
Ausg[mon] := 25.70;
Ausg[die] := 12.65;
usw.
```

Vorerst geschafft!

In der nächsten Ausgabe der Rest von Teil 5

Die Gewinner

„Rien ne va plus!" Mit diesem Ausspruch endet die Spielrunde der Computronic 4/87. Lustige Geschichten waren gefragt, folgendes kam bei raus:
Aus einem Aufsatz: Als Gretel die Hexe gegrillt hatte, lief sie erregt zum Hänsel und öffnete sein Ställchen.
Peter Hoekamp, 4971 Hüllhorst

Aus der Schule geplaudert: Unsere Physiklehrerin nahm sich vor, uns in einer Lehrprobe den Freien Fall zu demonstrieren. Das Treppenhaus unserer Schule war dafür ein geeigneter Versuchsort. Wir standen nun in der vierten Etage und wollten abstoppen, wie lange ein Stein braucht, um von hier auf den Kellerboden zu fallen. Unsere Lehrerin hielt also in einer Hand die Stoppuhr und in der anderen den Stein. Sie rief nun: „Eins, zwei, drei!", ließ die Stoppuhr los und drückte auf den Stein. Die Lehrprobe war natürlich gelaufen.
Nach einigen Sekunden war zu vernehmen, wie die Stoppuhr geräuschvoll ihr Ziel erreichte.
Gunnar Brandenburg, 1000 Berlin 30

Altersbedingt: Als meine Oma zum ersten Mal in ihrem Leben in ein Auto stieg, war sie schon 82 Jahre alt. Ich fuhr also mit ihr los. Ängstlich beobachtete sie, wie ich in andere Gänge schaltete. Erst nach mehreren Kilometern wagte sie es, mich anzusprechen: „Mädchen, achte Du lieber auf die Straße, das Benzin kann ich ja umrühren."
Ich bog mich vor Lachen und mußte auf dem nächsten Parkplatz halten. Das war das erste und letzte Mal, daß meine Oma im Auto gefahren ist.
Heidrun Albrecht, 2847 Barnstorf

Die Kirche als Schauplatz: Neulich saß ich in der Kirche, da hat sich doch tatsächlich eine Oma neben mir eine Zigarette angezündet. Da ist mir doch glatt das Bier aus der Hand gefallen.
Birgo Meier, 4459 Enlichheim

Fremdsprachen: Was heißt Condom auf niederländisch? – Antwort: Pipi Langstrumpf!
Martin Goerke, 2104 Hamburg 92

Der Wettbewerb

Echt klasse, Herr Schröder! Diesmal hat sich die Redaktion wieder etwas ganz Gemeines ausgedacht. In dieser Ausgabe muß ein „Logical" gelöst werden. Ihr müßt Martina, Manfred, Bernd, Philipp und Thomas jeweils die entsprechende Zigarettensorte, Automarke und deren Lieblingstier zuordnen.

Zigarettensorten: West, Marlboro, Benson & Hedges, Vannelle, Camel.

Automarken: Talbot, Suzuki, Ford, Audi, Opel

Lieblingstiere: Panda, Elefant, Papagei, Hund, Wildschwein.

Folgende Hinweise gibt die Redaktion:
1. **Bernd raucht Vannelle und mag keine Papageien.**
2. **Der Suzuki-Fahrer liebt Papageien.**
3. **Martina fährt keinen Opel und mag Elefanten.**
4. **Der Benson 6 Hedges-Raucher mag Wildschweine nicht und fährt Talbot.**
5. **Der Ford-Fahrer ist auf den Hund gekommen und raucht nicht Camel.**
6. **Der Marlboro-Raucher hat einen Papagei.**
7. **Der Talbot-Fahrer ist ein Panda-Fan und heißt Manfred.**
8. **Philipp raucht Camel und mag keine Papageien.**

Natürlich gibt es wieder fünf Programme aus dem Repertoire des Tronic-Verlags zu gewinnen?

Jetzt laßt einmal Euren Scharfsinn spielen. Mit ein wenig Geduld kommt Ihr sicher auf die Lösung. Viel Spaß wünscht die Redaktion.

Turbo-Pascal-Kurs (Fortsetzung Teil 5)

Der Mengentyp (SET)

Eine Menge (SET) ist eine Zusammenfassung mehrerer Obekte des selben Typs. Jedes einzelne Obekt einer Menge nennt man Element.

Regel: SET OF Basistyp

Der Basistyp muß ein einfacher Typ sein (ohne REAL) (z.B. auch Aufzählungstypen). Weiter muß beachtet werden, daß in einer Menge maximal 256 Elemente sind und die Ordnungszahl (ord(Variable)) der Elemente darf nicht größer als 255 sein (z.B. darf die INTEGER-Zahl 500 nicht drin sein, weil ord(500) = 500 ist).

Beispiele:

```
TYPE
Buchstaben  =SET OF 'A' :a 'Z';
Ziffern     =SET OF '0' :a '9';
Wochentage  =SET OF (mon,die,mit,
                     don,fre,sam,son);
gross_und_kleinbuchstaben =
SET OF ('A' :a 'Z','a' :a 'z');
```

Eine Menge kann auch lediglich innerhalb eines Programms angegeben werden. Dann werden die Elemente in eckigen Klammern aufgeführt.

Das Ganze hat etwas Ähnlichkeit mit dem ARRAY. Aber die Mengen unterscheiden sich deutlich vom ARRAY.
1. Es gibt keine Rangordnung. Die Mengen [1,3,5], [1,5,3] und [5,3,1] sind gleich.
2. Jedes Element ist in einer Menge nur einmal vertreten. Die Mengen [1,3,5,1] und [1,3,5] sind deshalb ebenfalls gleich.
3. Eine Menge kann nur als Ganzes angesprochen werden. Ein Zugriff auf einzelne Elemente mit einem Index (wie beim ARRAY) ist also nicht möglich.

Was im einzelnen mit diesem Mengentyp gemacht werden kann, können wir hier nicht darstellen, weil dies den Rahmen eines Einführungskurses sprengen würde. Eine Anwendung wollen wir jedoch zeigen, für die sich der Mengentyp hervorragend eignet.

Aufgabe: Es soll von der Tastatur ein Zeichen eingelesen und jeweils entschieden werden, ob es ein Vokal ist oder nicht.

Das Programm könnte folgendermaßen aussehen:

```
1:  PROGRAM Vokaltest;
2:  VAR zeichen: char;
3:  BEGIN
4:     clrscr;
5:     REPEAT
```

```
6:      write('Bitte ein Zeichen eingeben
        („0“ fuer Ende): ');
7:        readln(zeichen);
8:      IF (zeichen='A') OR (zeichen=
        'E') OR (zeichen='I') OR
9:      (zeichen='O') OR (zeichen='U')
        OR (zeichen='a') OR
10:     (zeichen='e') OR (zeichen='i')
        OR (zeichen='o') OR
11:     (zeichen='u')
12:    THEN writeln(zeichen,' ist ein
       Vokal')
13:    ELSE writeln(zeichen,' ist kein
       Vokal');
14:    writeln;
15:  UNTIL zeichen='0'
16: END.
```

Über die IF-Bedingung braucht man wohl nicht viel zu sagen. Man kann sie natürlich verstehen, wenn man sie in Ruhe durchliest, aber übersichtlich ist sie bestimmt nicht. Deshalb gleich ein Programm, in dem die IF-Bedingung mit Hilfe einer Menge formuliert wird.

```
1:  PROGRAM Vokaltest-2;
2:  VAR zeichen : char;
3:  BEGIN
4:      clrscr;
5:      REPEAT
6:      write('Bitte ein Zeichen eingeben
        („0“ fuer Ende): ');
7:      readln(zeichen);
8:      IF zeichen IN ['A','E','I','O','U',
        'a','e','i','o','u']
9:      THEN writeln(zeichen,' ist ein
        Vokal')
10:     ELSE writeln(zeichen,' ist kein
        Vokal');
11:     writeln;
12:   UNTIL zeichen = '0'
13: END.
```

Diese Verbesserung spricht wohl für sich selbst. Ihr seht bei diesem Beispiel auch, daß man in einem Programm eine Menge angeben kann. Die Regel (Syntax) für die Abfrage, ob ein bestimmtes Zeichen in einer Menge enthalten ist, lautet:
bestimmtes_Zeichen IN Menge
In unserem Beispiel wurde diese Abfrage in eine IF-Anweisung gesteckt. (Bei einer WHILE-Schleife würde so eine Mengenabfrage folgendermaßen aussehen:
WHILE best_Zeichen IN Menge DO)

Gleich noch ein kurzes Beispiel für die übersichtliche Abfrage mit Mengen. Es soll beispielsweise geprüft werden, ob ein Zeichen eine Ziffer ist. Die Bedingung ohne Menge wäre:
IF (zeichen >='0') AND (zeichen <='9') THEN . . .

Die Alternative mit einer Menge:
IF zeichen IN ['0' . . '9'] THEN . . .

Der RECORD-Typ

Die Datentypen, die du bisher kennenlernt hast, und die aus mehreren Elemente bestehen (ARRAY, Menge, STRING) hatten gemeinsam, daß die einzelnen Elemente den gleichen Datentyp haben müssen. Zum Beispiel kann ein ARRAY nur Elemente vom Typ INTEGER oder nur Elemente vom Typ CHAR haben. Mit dem RECORD-Typ ist es möglich, Elemente zusammenzufassen zu einem Datentyp, die aus verschiedenen Datentypen bestehen.
Nehmen wir als Beispiel eine Adressendatei, die folgendermaßen aufgebaut ist: Die einzelnen Adressen sind auf Karteikarten geschrieben und in einem Karteikasten untergebracht. Auf einer Karteikarte ist jeweils eine Adresse untergebracht. Jede Adresse besteht aus folgenden Informationen:
Name, Vorname, Straße, Hausnummer, Postleitzahl, Ort, Telefon

Eine solche Karteikarte bezeichnet man auch als einen Datensatz (aus der gesamten Datei, die im Karteikasten ist). Diese Einheit eines Datensatzes kann nun in Pascal ausgedruckt werden, indem man einen RECORD definiert, der aus den oben aufgeführten Bestandteilen besteht. Damit ist dann auch verständlich, warum der RECORD auch als Datensatz bzw. als Verbund bezeichnet wird.

```
TYPE AdrKartei = RECORD
Name     :  STRING [25];
Vorname  :  STRING [15];
Strasse  :  STRING [25];
Hausnr   :  1..1000;
PLZ      :  1000..9999;
Ort      :  STRING [25];
Telefon  :  STRING [10]
END;
```

Die RECORD-Definition beginnt also mit dem reservierten Wort RECORD und endet mit END. Dazwischen werden die einzelnen Bestandteile so definiert, wie du das aus der Variablen-Deklaration schon kennst. Auch können die bekannten Datentypen verwendet werden (z. B. ARRAY, SET etc.).
Auf jedes einzelne Element des RECORDS kann zugegriffen werden. Dabei gibt es eine bestimmte Regel, die wir gleich mal demonstrieren, indem ein Datensatz mit Werten gefüllt wird. Die Anweisung dafür sieht folgendermaßen aus:

(Im Deklarationsteil muß eine Variable „Adresse“ vom Typ „AdrKartei“ definiert worden sein: VAR Adresse: AdrKartei)

```
BEGINN
Adresse.Name     : = 'Mueller';
Adresse.Vorname  : = 'Hans';
Adresse.Strasse  : = 'Hauptstrasse';
Adresse.Hausnr   : = 13;
Adresse.PLZ      : = 1000;
Adresse.Ort      : = 'Berlin';
Adresse.Telefon  : = '123 45 67'
END.
```

Der Zugriff auf ein Feld eines RECORDS geschieht also nach folgender Regel:
Name_des_Records . Name_des_Feldes

„Ganz schön umständlich“, wirst du sagen. „Und viel Tipparbeit“, allein beim Ausfallen unseres Datensatzes. Damit hast du recht. Diese Arbeit kann kan jedoch erheblich verringern. Wir werden das gleich am Beispiel zeigen:

```
BEGINN
  WITH Adresse DO
    Beginn
      Name     : = 'Mueller';
      Vorname  : = 'Hans';
      Strasse  : = 'Hauptstrasse';
      Hausnr   : = 13;
      PLZ      : = 1000;
      Ort      : = 'Berlin';
      Telefon  : = '123 45 67'
    END
END.
```

Zu Beginn der Zuweisungen passiert folgendes: Dem Computer wird „gesagt“: Mit (WITH) dem Record „Adresse“ mache folgendes (DO). Jetzt „weiß“ der Rechner, daß die Variable „Name“ zum Record „Adresse“ gehört usw.
Es mag sein, daß dir das ganze sehr kompliziert vorkommt und du überhaupt nicht verstehst, worin der Vorteil eines Records liegen soll, wenn alle Elemente dann doch einzeln verarbeitet werden müssen. Dazu kurz zwei Anmerkungen: 1. Der Vorteil des Records liegt wieder in der Programmklarheit. Man sieht nämlich sofort, zu welcher Einheit ein einzelnes Element gehört (oft hat man nämlich mehr als einen Record).
2. Ein wesentlicher Vorteil des Record wird dir erst klar, wenn du Funktionen und Prozeduren kennengelernt hast und Verarbeitung von Files. Hier ist es nämlich möglich, einen Record in „einem Stack“ zur Weiterverarbeitung zu übergeben. Da wird dann der Vorteil sehr schnell deutlich. Also noch ein bißchen Geduld.

Ende Teil 5

Turbo Pascal Kurs Teil 6

In diesem Teil werden wir das Inventar von Werkzeugen, das wir bisher kennengelernt haben, um Programme zu strukturieren, wesentlich erweitern: Es geht diesmal um sogenannte Unterprogramme. Unterprogramme sind, um es kurz vorwegzunehmen, Teilprogramme, die solche Aufgaben erledigen, die immer wiederkehren. In PASCAL werden Unterprogramme entweder Funktionen oder Prozeduren genannt. Wer schon mit BASIC programmiert hat, wird das Kommando GOSUB .. RETURN kennen; damit kann eine ähnliche Struktur erreicht werden - am besten ist aber, daß man alle BASIC-Kenntnisse vergißt, wenn man den Charakter von PASCAL-Prozeduren wirklich verstehen will.

Viele Befehle, die PASCAL kennt (die also vordefiniert sind), sind nichts anderes, als Funktionen oder Prozeduren. Eine vordefinierte Funktion ist z.B. bsqr(x)b:. Diese Funktion berechnet das Quadrat einer beliebgen Zahl x. Man sieht, daß viele mathematische Funktionen in PASCAL als fertige „Funktionen" eingebaut sind, aber beileibe nicht alle. Deshalb gibt es die Möglichkeit, beliebige Funktionen zu definieren und im Programmablauf zu benutzen, als ob es vordefinierte Funktionen (oder Prozeduren) wären. Was eine Funktion von einer Prozedur unterscheidet, werdet Ihr gleich erfahren, zunächst zeige ich mal ein kleines Beispiel.

In vielen Programmen ist es sehr nützlich, den Programmablauf an bestimmten Stellen anzuhalten und es dem jeweiligen Benutzer zu überlassen, wann das Programm seine Tätigkeit fortsetzt. In den vorherigen Teilen dieses Kurses haben wir schon einige Male den Ausdruck REPEAT UNTIL KEYPRESSED verwendet, um ein Programm anzuhalten und die Fortsetzung von einem beliebigen Tastendruck abhängig zu machen. Wenn solche Stellen im Programmtext häufiger vorkommen, ist es sehr sinnvoll, eine Prozedur zu definieren, die folgendes macht:

1. Anhalten des Programmablaufs;
2. Ausgeben einer Meldung an den Benutzer, daß durch Drücken einer beliebigen Taste das Programm fortgesetzt werden kann.

Eine Prozedur wird im Deklarationsteil definiert, und zwar anschließend an die Variablendefinition. Jede Prozedur ist im Prinzip ein eigenes Miniprogramm, das mit BEGIN beginnt und mit END endet und einen eigenen Namen hat.

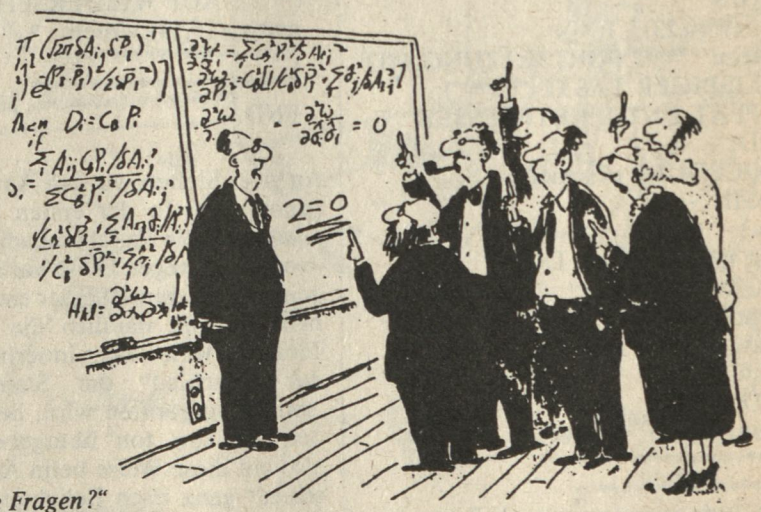
Folgendes Programm soll Zahlen von 1 bis 10 ausdrucken und nach jedem Ausdruck anhalten (es ist ein ziemlich sinnloses Programm und soll nur die Benutzung der Prozedur demonstrieren).

```
 1: PROGRAM prozedur-demo;
 2: VAR i: integer;
 3: PROCEDURE stop;
 4: BEGIN
 5: gotoxy(10,23);
 6: writeln('****** FORTSETZUNG
    MIT BELIEBIGER TASTE ******');
 7: REPEAT UNTIL KEYPRESSED
 8: END;
 9: BEGIN ä Hauptprogrammü
10: clrscr;
11: FOR i:=1 to 10 DO
12: BEGIN
13: gotoxy(40,10);
14: writeln(i);
15: stop
16: END
17: END.
```

Was hier passiert, ist leicht zu verstehen. Immer wenn im Hauptprogramm der Name der Prozedur „stop" erscheint, wird genau das gemacht, was im Deklarationsteil unter der Prozedur „stop" definiert wurde. Man erspart sich also durch die Benutzung von Prozeduren die mühselige Arbeit, immer wiederkehrende Kommandofolgen im Hauptprogramm immer aufs Neue einzutippen, sondern man macht das einmal am Anfang und ruft diese Kommandofolge einfach mit dem Prozedurnamen auf. Auf diese Weise kann man u.U. viele Duzend Zeilen Programmtext durch Nennung eines kurzen Namens ausfallen lassen.

Zusätzlich haben wir im obigen Programm eine neue Standardprozedur von TURBO-PASCAL kennengelernt: gotoxy(x,y). Damit kann man den Cursor an eine bestimmte Bildschirmstelle bringen und so z.B. einen Text dort ausdrucken lassen. Die Prozedur gotoxy bekommt zwei Zahlenwerte mit auf den Weg: der erste Wert in der Klammer gibt an, in welche Spalte der Cursor soll, der zweite Wert in der Klammer gibt an, in welche Zeile der Cursor soll. In der Regel haben wir Monitore mit 80 Spalten und 25 Zeilen, so daß der erste Wert von 1 bis 80, der zweite von 1 bis 25 gehen kann. Unsere Prozedur hat also die Standardprozedur gotoxy benutzt, d.h. man kann innerhalb von Prozeduren andere Prozeduren benutzen (vorausgesetzt, diese sind bereits definiert).

Zur Übung noch eine kleine Prozedur, die man ans Ende von Programmen hängen kann:



„Irgendwelche Fragen?"

```
PROCEDURE abschied;
BEGIN
clrscr;
FOR i:=3 downto 1 DO
BEGIN
sound(i*200);
delay(1000);
nosound
END;
gotoxy(15,12);
writeln('******************
*******************');
gotoxy(15,13);writeln
```

```
('*      AUF WIEDERSEHEN      *');
gotoxy(15,14);writeln
('**********************
******************');
END.
```

Die Standardprozeduren sound, nosound und delay dienen hier zur Tonerzeugung (vorausgesetzt, Euer Computer hat einen eingebauten Lautsprecher, sonst funktioniert dieser Teil natürlich nicht). „Sound" erzeugt einen Ton mit der Frequenz, die in Klammern angegeben ist, und zwar solange, bis der Ton mit „Nosound" abgeschaltet wird. Die Zeitdauer kann man mit „Delay" regulieren: delay macht eine Pause im Programmablauf, die durch die Zahl in Klammern bestimmt werden kann, und zwar in Tausendstel-Sekunden.

Hängt diese Prozedur zwischen Zeile 16 und 17 an das erste Beispielprogramm an, compiliert das Ganze, und seht Euch an, was passiert. Je nach Geschmack kann man solche „Abschiedsprozeduren" noch mehr ausschmücken oder zusätzlich „Begrüßungsprozeduren" schreiben usw..
Das gesamte Programm sieht dann so aus:

```
 1: PROGRAM prozedur-demo;
 2: VAR i: integer;
 3: PROCEDURE stop;
 4: BEGIN
 5: gotoxy(10,23);
 6: writeln('****** FORTSETZUNG MIT
    BELIEBIGER TASTE ******');
 7: REPEAT UNTIL KEYPRESSED
 8: END;
 9: PROCEDURE abschied;
10: BEGIN
11: clrscr;
12: FOR i:=3 downto 1 DO
13: BEGIN
14: sound(i*200);
15: delay(1000);
16: nosound
17: END;
18: gotoxy(15,12); writeln
('**********************
**********************');
19: gotoxy(15,13);writeln
('*    AUF      WIEDERSEHEN
20: *');
21: gotoxy(15,14);writeln
('**********************
**********************');
22: END.
23:
24: BEGIN ä Hauptprogrammü
25: clrscr;
26: FOR i:=1 to 10 DO
27: BEGIN
28: gotoxy(40,10);
29: writeln(i);
30: stop
31: END;
32: abschied
33: END.
```

An den vordefinierten Prozeduren habt Ihr schon erkannt, daß man Prozeduren nicht nur einfach mit ihrem Namen aufrufen, sondern ihnen in Klammern bestimmte Werte mit auf den Weg geben kann, die das jeweilige Verhalten der Prozeduren beeinflussen. Solche Werte nennt man Parameter, und sie müssen selbstverständlich im Deklarationsteil beim Definieren der Prozedur angegeben werden. Man könnte die Prozedur „abschied" folgendermaßen ändern:

```
- PROCEDURE abschied(ton: integer);
  BEGIN
  clrscr;
  FOR i:=3 downto 1 DO
  BEGIN
- sound(i*ton);
  delay(1000);
  nosound
  END;
  gotoxy(15,12);writeln
  ('**********************
  ********************');
  gotoxy(15,13);writeln
  ('*    AUF WIEDERSEHEN      *');
  gotoxy(15,14);writeln
  ('********************
  **********************');
  END.
```

An zwei Stellen wurden Änderungen vorgenommen: In der ersten Zeile kam in Klammern ein Variablenname hinzu: - ton - vom Typ integer. Diese Variable ist ein sogenannter Parameter, der einen Wert übernehmen soll, nämlich die beabsichtigte Tonhöhe. Wenn dann innerhalb der Prozedur „abschied" die Standardprozedur „sound" aufgerufen wird, bekommt diese den Wert von „ton" übergeben. Man kann also auf diese Weise beim Aufruf von „abschied" ganz nach Belieben die Tonhöhe der Abschiedsmelodie bestimmen. Was hier wie Spielerei klingt, ist aber ein sehr wichtiger Vorgang, nämlich die Übergabe von Parametern an Prozeduren. In diesem Vorgang stecken noch eine Menge Geheimnisse, auf die wir in der nächsten Folge vertieft eingehen werden.

Was unterscheidet nun Funktionen von Prozeduren? Der wesentliche Unterschied kann folgendermaßen beschrieben werden: Bei Funktionen kann es mehrere Parameter geben, aber nur einen Ergebniswert, der dem Funktionsnamen zugewiesen wird. Bei Prozeduren kann es ebenfalls mehrere Parameter geben und ebenfalls mehrere Ergebniswerte, die nicht dem Prozedurnamen, sondern den übergebenen Variablen zugewiesen werden. Bei Funktionen dient der Funktionsname praktisch als Variablenname, der einen Wert zugewiesen bekommt, bei Prozeduren dient der Name lediglich zur Definition und zum Aufruf der Prozedur.
Ein Beispiel für eine Funktion soll dies verdeutlichen:
Aus dem Mathematikunterricht kennt Ihr vielleicht noch den Begriff „Fakultät". Die Fakultät ist wie „Quadrat" die Funktion einer Zahl, und zwar ist z.B. 'Fakultät von 4' = 1 mal 2 mal 3 mal 4. Man multipliziert also alle Zahlen bis 4 (oder bis zu der Zahl, von der man die Fakultät wissen will). Die Abkürzung ist das Ausrufezeichen: 4!=24. Noch ein Beispiel: 6! = 1 mal 2 mal 3 mal 4 mal 5 mal 6 = 720; viele Taschenrechner haben diese Funktion.
PASCAL hat diese Funktion aber nicht - also müssen wir sie definieren.
Das geschieht im Definitionsteil:

```
1: FUNCTION fakultaet(x:integer):
   integer;
2: VAR i,resultat: integer;
3: BEGIN
4: resultat:=1;
5: FOR i:=1 TO x DO
6: resultat:=resultat*i;
7: fakultaet:=resultat
8: END;
```

Die Definition beginnt mit dem Schlüsselwort FUNCTION: jetzt weiß PASCAL, daß der folgende Name der Name einer vom Programmierer definierten Funktion ist. In der Klammer folgt ein Variablenname (hier x) mit Typ-Angabe: diese Variable ist die Variable, die „übergeben" wird. Wenn wir z.B. sqr(5) schreiben, „übergeben" wir an die Funktion sqr den Wert 5 (wir könnten stattdessen natürlich auch eine Variable verwenden, die einen Wert hat). Mit dieser Variable x wird nun innerhalb der Funktion gerechnet, nämlich wie üblich zwischen BEGIN und END. Wie das hier funktioniert, sollte leicht nachvollziehbar sein. Wichtig ist nun folgendes: Das Ergebnis der Berechnung (eben die Fakultät einer Zahl) wird dem Funktionsnamen zugewiesen! Das geschieht hier in Zeile 7. Der Funktionsname ist also gewissermaßen selbst der Name einer Variable, die einen Wert haben kann - eben den Wert, der innerhalb der Funktion ausgerechnet wurde. Deshalb muß auch die Funktion einen Typ zugewiesen bekommen (in Zeile 1 bekam die Funktion fakultät den Typ integer). Ist

die Funktion so definiert, kann sie im weiteren Programmverlauf beliebig oft verwendet werden, z.B.

```
 1: PROGRAM fakultaet;
 2: VAR zahl: integer;
 3: FUNCTION fakultaet(x:integer):
   : integer;
 4: VAR i: integer;
 5: resultat: integer;
 6: BEGIN
 7: resultat:=1;
 8: FOR i:=1 TO x DO
 9: resultat:=resultat*i;
10: fakultaet:=resultat
11: END;
12: BEGIN
13: writeln('Eingabe einer Zahl: ');
14: readln(zahl);
15: IF zahl  8 THEN writeln
   : (fakultaet(zahl))
16: ELSE BEGIN
17: writeln('Integer-Zahlen über sieben
   : können ');
18: writeln('nicht berechnet werden!')
19: END
20: END.
```

Warum in den Zeilen 17 und 18 behauptet wird, daß die Fakultät von Zahlen über 7 nicht berechnet werden kann, versucht Euch selbst zu überlegen – und wie man Abhilfe schaffen könnte (Lösung im nächsten Teil).

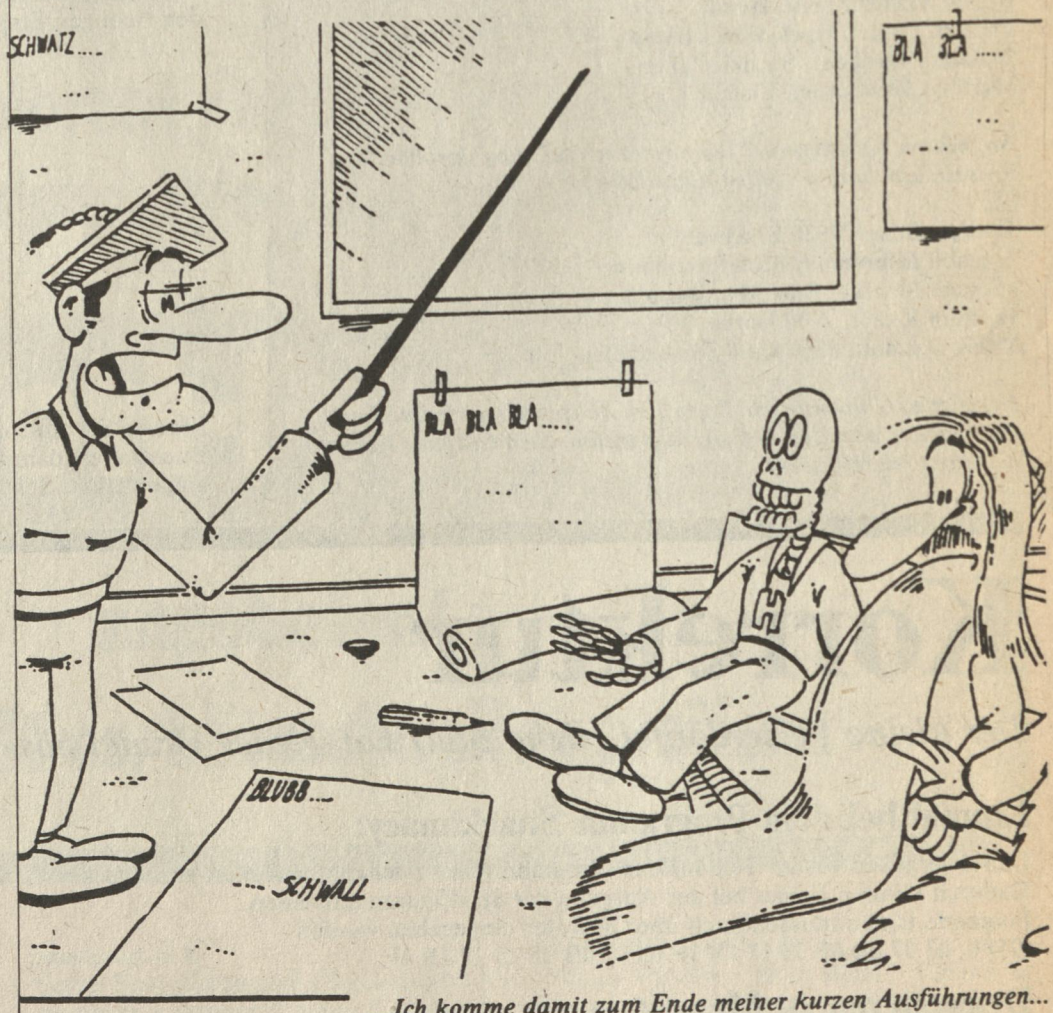
Hat man einmal so eine Funktion geschrieben, dann empfiehlt es sich, das entsprechende Programmstack abzuspeichern und aufzuheben. Denn es wäre ja Unsinn, in jedem neuen Programm, in dem man die entsprechende Funktion benötigt, sie neu zu schreiben. Auf diese Weise kann man sich mit der Zeit eine umfangreiche Funktionen- und Prozeduren-Bibliothek anlegen, aus der man je nach Bedarf die benötigten Teile in sein Programm einbaut.

Für diesen Vorgang des Einbaus fertiger Funktionen oder Prozeduren in neue Programme gibt es in TURBO-PASCAL eine elegante Möglichkeit: den sogenannten Include-Compilerbefehl. Compilerbefehle sind Befehle, die nur im Moment der Compilierung (wenn man die C-Taste gedrückt hat) von Bedeutung sind und dem Compiler bestimmte Anweisungen geben. Erscheint am Anfang eines Programms z.B. folgende Zeile:

ä$I fakult.incü

so bedeutet das für den Compiler, daß er auf der Diskette nach einer Datei mit dem Namen „fakult.inc" sucht, diese Datei lädt und in das gerade zu compilierende Programm einbaut, als ob man sie direkt an diese Stelle geschrieben hätte. Die allgemeine Regel lautet: Innerhalb der Kommentarklammer kommt die Zeichenfolge $I und anschließend der Dateiname (existiert die Datei nicht, gibt es einen Fehler). Versucht also folgendes: Speichert die Funktion „fakultät" (sonst nichts, also nicht das ganze Programm) auf der Arbeitsdiskette unter dem Namen „fakultaet.inc" ab. Dann schreibt das Programm „Fakultaet" so um:

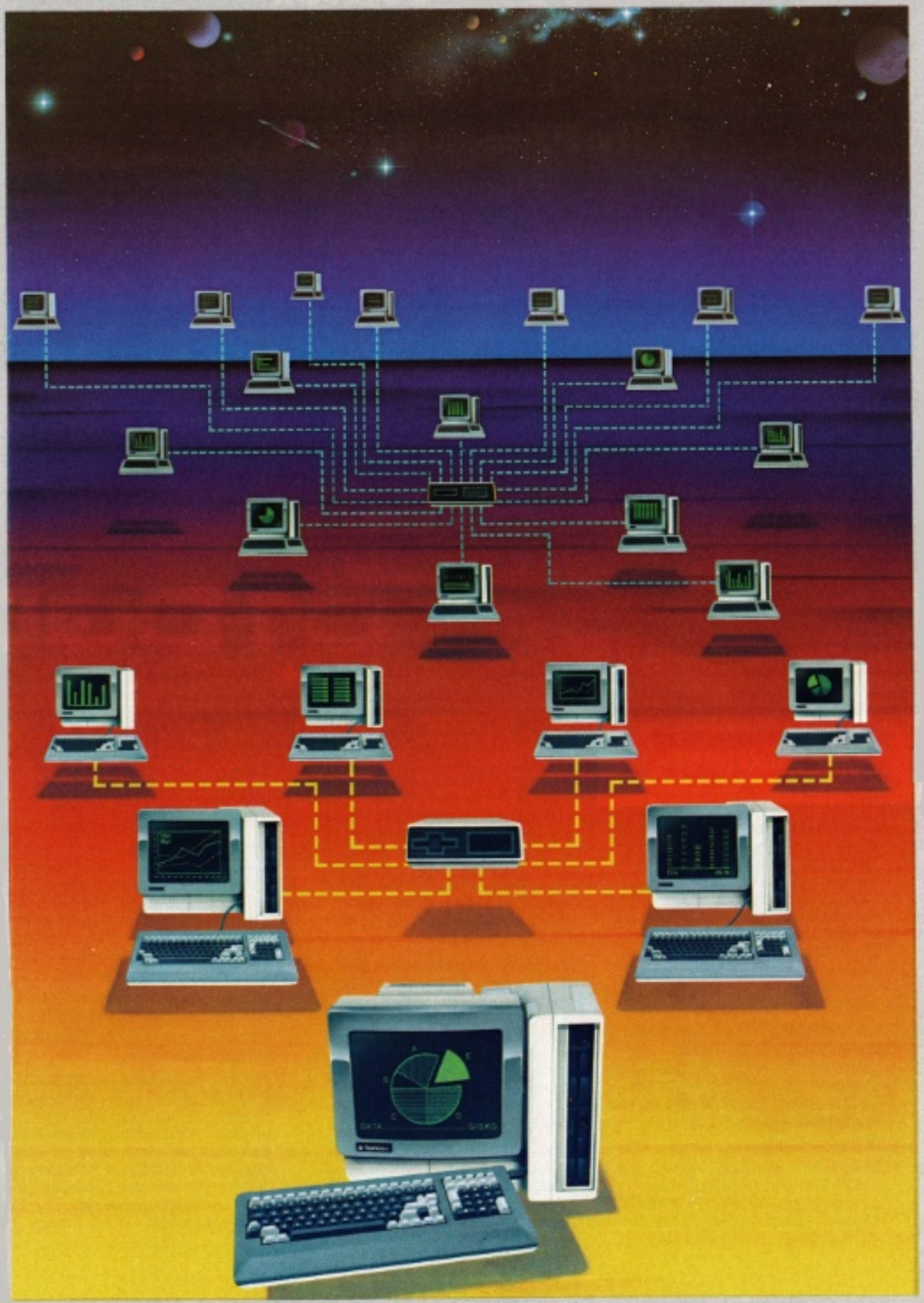
```
 1: PROGRAM fakultaet;
 2: VAR zahl: integer;
 3:
 4: ä$I fakultaet.incü
 5:
 6:
 7:
 8:
 9:
10:
11:
12: BEGIN
13: writeln('Eingabe einer Zahl: ');
14: readln(zahl);
15: IF zahl  8 THEN writeln(fakultaet
    (zahl))
16: ELSE BEGIN
17: writeln('Integer-Zahlen aber sieben
können ');
18: writeln('nicht berechnet werden!')
19: END
20: END.
```



„...Ich komme damit zum Ende meiner kurzen Ausführungen...

Jetzt compiliert das Programm, und Ihr werdet sehen, daß der Compiler automatisch auf der Diskette nach „fakultät.inc" sucht, diese Datei lädt und das Programm so funktioniert, als hättet Ihr die Funktion direkt in den Programmtext hineingeschrieben.

Soviel für diesmal. Nächstes mal geht's weiter mit Funktionen und Prozeduren, denn dieser Teil war nur ein kleiner Vorgeschmack auf die Möglichkeiten von Prozeduren und Funktionen.



This collection is for all TI99/4A user. If you like TI99 computer join at international forum present on AtariAge.com site.

Greetings from Italy. ***Astharot***